

12017IRI

GPO PRICE \$ _____

CFSTI PRICE(S) \$ _____

Contract NAS 12-38

Hard copy (HC) 3.00

Microfiche (MF) 1.00

653 July 65

INTERIM REPORT

SPACEBORNE MEMORY ORGANIZATION

To NASA/ERC

FACILITY FORM 602	N66 37280	
	(ACCESSION NUMBER)	(THRU)
	<u>121</u> (PAGES)	<u>1</u> (CODE)
	<u>CR-28278</u> (NASA CR OR TMX OR AD NUMBER)	<u>08</u> (CATEGORY)

HONEYWELL SYSTEMS & RESEARCH DIVISION

RQT38945-A

Interim Report

SPACEBORNE MEMORY ORGANIZATION

Contract NAS 12-38

HONEYWELL Systems & Research Division

ACKNOWLEDGMENTS AND APPROVAL

HONEYWELL REPORT 12017FR1

Prepared by:

D. C. Gunderson

C. W. Hastings

G. J. Prom

Approved by: O. Hugo Schuck / *AS*

O. Hugo Schuck
Director of Research

CONTENTS

	Page
SECTION I INTRODUCTION	1-1
SECTION II SUMMARY OF SPACE EXPLORATION COMPUTATION REQUIREMENTS	2-1
Computation Descriptors	2-1
Conclusions	2-6
SECTION III A SURVEY OF ASSOCIATIVE MEMORY ORGANIZATIONAL APPROACHES	3-1
Introduction	3-1
Basic Searches	3-3
Equality Search	3-3
Inequality Search	3-4
Maximum (Minimum) Search	3-4
Proximity Search	3-5
Intersection of Searches	3-5
Union of Searches	3-6
Processing Operations	3-6
Addition ($X_1 + S \dots$)	3-6
Field Addition ($X_1 + Y_1 \dots$)	3-7
Summation ($X_1 + X_2 \dots$)	3-7
Counting ($X_1 + 1 \dots$)	3-7
Shifting	3-7
Complementing ($X_1 \dots$)	3-7
Logical Sum $-(X_1 \dots)$	3-8
Logical Product ($X_1 \dots$)	3-8
Floating Point Add	3-8
Facilities for Storage and Processing of Results	3-10
Shift Capability	3-11
Logic to Determine if at Least One Match	3-12
Logic to Determine if Exactly One Match	3-13
Logic to Determine Exact Number of Matches	3-13
Word Select Ladder	3-14
Organization Approaches	3-15
Approach 1 - Minimum Associative Memory	3-15
Approach 2 - Associative Memory with Bit Slice Writing	3-30
Approach 3 - Associative Memory with All-Parallel Equality Search	3-43
Approach 4 - Local Logic - Ternary Output	3-54
Approach 5 - Intercommunicating Cells	3-66

CONTENTS

	Page
Summary of Organizational Approaches	3-72
Bibliography for Section III	3-75
Magnetic Associative Memories	3-75
Cyrogenic Associative Memories	3-75
Semiconductor Associative Memories	3-76
Associative Memory Systems	3-77
Associative Memory Algorithms	3-78
Applications of Associative Memories	3-79
Associative Processors	3-80
References for Section III	3-81
SECTION IV	
INVESTIGATION OF SPECIAL APPLICATIONS AND APPROACHES	4-1
An Associative Memory for Incremental Computation	4-2
Some Considerations in the Organization of an Ultra-Reliable Associative Memory	4-12
SECTION V	
SUMMARY OF RESULTS AND WORK TO BE DONE	5-1

ILLUSTRATIONS

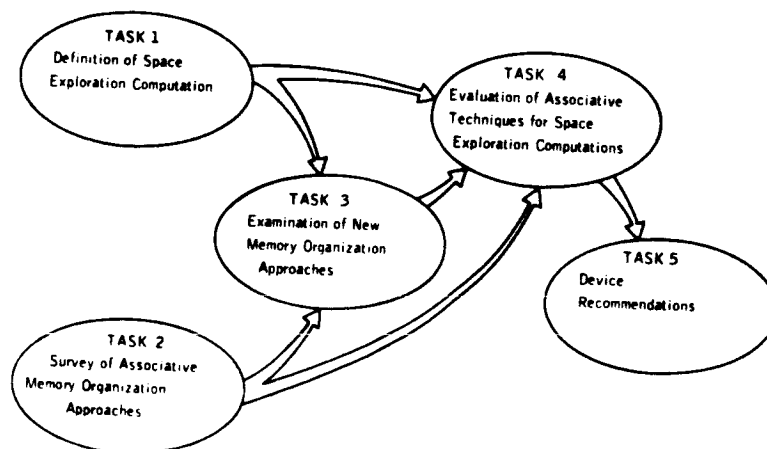
Figure		Page
3-1	Functional Block Diagram	3-2
3-2	Approach 1 - Minimum Associative Memory	3-17
3-3	Transpolarized Memory With Two Dimensional Readout	3-26
3-4	Transpolarizer NDRO Readout	3-27
3-5	Approach 2 - Associative Memory With Bit Slice Writing	3-31
3-6	Approach 3 - Associative Memory With All-Parallel Equality Search	3-44
3-7	Approach 4 - Associative Memory With Local Logic - Ternary Output	3-56
3-8	Approach 5 - Associative Memory With Intercommunicating Cells	3-68
4-1	Block Diagram of an Integrator for an Incremental Computer	4-3
4-2	Memory Arrangement for a Bit Slice Associative Memory Used For Incremental Computation	4-6
4-3	Plugboard Programming Arrangement For a Bit Slice Associative Memory Used For Incremental Computation	4-9
4-4	Memory Arrangement For a Word Slice Associative Memory Used for Incremental Computation	4-11

SECTION I INTRODUCTION

This report contains the results of the first five months of effort on a study aimed at the investigation of the use of associative memories in unmanned space vehicles in the 1975 to 1985 time period. It is assumed that unmanned planetary exploration will require computation of a type and extent not previously provided in aero-space vehicles. This is partially due to the fact that the vehicle is operating in an unknown environment with a limited earth-vehicle communications link.

Associative memories with their search capabilities have in the past been proposed primarily for non-numerical tasks such as information retrieval. However, associative memories capable of processing operations as well as search operations can be applied effectively to a much wider class of problems. They are applicable to problems of the type requiring not only table look-up operations, but also processing operations of the type where the same operation is being carried out over a large set of operands. It is felt that these capabilities justify the investigation of associative techniques for use in a sophisticated on-board computing facility.

The study is divided into five tasks as shown in the diagram below:



Work to date has been on the first three tasks of the study. Tasks 1 and 2 are completed and Task 3 is partially completed.

The results of Task 1, the Definition of the Space Exploration Computation Requirements, are presented in Appendix A. A summary of these results is contained in Section II of the report. Four of the major computational functions have been defined and a number of conclusions regarding fruitful areas of application of associative techniques are given.

Task 2 was performed as two essentially independent subtasks. One was a survey of associative memory organizational approaches, and the other was a survey of devices suitable for use in associative memories. The organizational approach survey is included as Section III of the report. Definitions concerning both the organization and capabilities of an associative memory are given. Five basic organizational approaches along with a number of variations are described in terms of their capabilities to perform search and processing operations. The devices suitable for mechanization of each of these approaches are discussed. The device survey subtask, is included as Appendix B of this report. This appendix contains a description of all devices with a memory capability felt to be suitable for use in future associative memories.

In Task 3, the results of Task 1 were evaluated and studied to determine promising areas of application and general capability requirements for associative memories. Five areas of applications have been singled out for further investigation. One of these, the use of an associative memory for incremental computations has been completed and is discussed in Section IV. Also included in Section IV is a brief discussion of some considerations in the organization of an ultra-reliable associative memory.

Section V contains a summary of the results obtained thus far and an outline of the work to be completed on the remainder of the study program.

SECTION II

SUMMARY OF SPACE EXPLORATION COMPUTATION REQUIREMENTS

To evaluate the application of associative memories to future planetary exploration with unmanned vehicles, it was necessary to define the on-board computation requirements. The time period of interest is from 1975 to 1985. While specific requirements are largely unknown at this time, general requirements were determined and potential problem areas were identified.

Details of this work are included as Appendix A. A summary of the requirements and a discussion of conclusions are contained within this section.

COMPUTATION DESCRIPTORS

The approach to defining computational requirements was to describe each in terms of a set of computation descriptors. The aim is to determine the form of the computation rather than the details. The attempt was to use computation descriptors which are at least one level higher than operations such as equality search, addition, multiplication, etc. A listing is contained in Tables 2-1 and 2-2.

Table 2-1

Functions	Descriptors
A. Data Handling 1. Data Acquisition a) Adaptive selection of scientific instruments b) Adaptive selection of performance inputs	 Simple decision making Simple decision making

Table 2-1 (Continued)

Functions	Descriptors
c) Adaptive sampling of scientific data	Simultaneous arithmetic Simultaneous comparisons
d) Adaptive sampling of performance data	Simultaneous arithmetic Simultaneous comparisons
e) System status monitoring	Simultaneous arithmetic Simultaneous comparisons
f) Dynamics range adjustment	
g) Conversion from analog to digital	
2. Storage and Related Control Requirements	
a) Data storage	
b) Storage allocation	Simultaneous comparisons
c) Data retrieval	Searching
3. Data Distribution	Simple Decision Making
B. Processing of Scientific Data	
1. Data compression	
a) Encoding methods	
1) Δ modulation	Matrix subtraction
2) Debiasing	Simultaneous comparisons Simultaneous arithmetic
3) Interval suppression	Simultaneous comparisons
4) Code substitution	Simultaneous comparisons
b) Filtering methods	
1) Statistical representation	
a) Quantiles	Simultaneous comparisons Sequential arithmetic

Table 2-1 (Continued)

Functions	Descriptors
<ul style="list-style-type: none"> b) Direct computation of moments c) Direct Approximation of the Probability Density Function 	<ul style="list-style-type: none"> Sum of Products Simultaneous arithmetic
<ul style="list-style-type: none"> 2) Curve fitting <ul style="list-style-type: none"> a) Interpolators b) Linear regression c) Least squares 	<ul style="list-style-type: none"> Polynomial evaluation Simultaneous arithmetic Simultaneous arithmetic Matrix inversion Sum of Products Matrix inversion
<ul style="list-style-type: none"> 3) Correlation 4) Feature oriented techniques 	<ul style="list-style-type: none"> Sum of Products Sum of Products
<ul style="list-style-type: none"> 2. On-board Decision Making <ul style="list-style-type: none"> a) Composition analysis b) Picture processing <ul style="list-style-type: none"> 1) Contrast comparisons 2) Operator techniques c) Other situations <ul style="list-style-type: none"> 1) Allocation of Resources 2) Task Scheduling 	<ul style="list-style-type: none"> Sum of Products Sequential arithmetic Sequential comparisons Simultaneous arithmetic Simultaneous comparisons Sum of Products
<ul style="list-style-type: none"> 3. Miscellaneous Data Processing <ul style="list-style-type: none"> a) Side looking radar 	<ul style="list-style-type: none"> Simultaneous arithmetic Sum of Products (4×10^6 multiplications/sec.)

Table 2-2

Function	Descriptor	Comments
A. Navigation of Spacecraft		
1. Reference Trajectory and State Transition Matrix Computations		
a) Direct Integration Method	Numerical Integration	6 equations Repeated every few seconds
b) Tabular Interpolation-extrapolation	Polynomial Evaluation	6 polynomials of 5th order Repeated every 5 to 10 minutes
c) Analytic Two-body Integrals	Formula Evaluation Matrix multiplication	≈ 20 formula 8 required Repeated every 5 to 10 minutes
2. State Estimation Techniques		
a) Kalman Filter	Matrix Multiplication Matrix Inversion Matrix Addition Matrix Transposition	6×6 , several per cycle $P \times P$, where $P = \text{no. of observations}$ 6×6 , $P \times P$ 6×6 , several per cycle (5 to 10 minutes repetition rate)
B. Navigation and Control of Lander		
1. Gyro signal corrections	Matrix multiplication (Incremental)	1 000 to 10, 000 per second [3 x 3] [3 x 1]
2. Accelerometer signal corrections	Matrix multiplication (Incremental) Formula evaluation (Incremental)	100 to 200 per second [3 x 3] [3 x 1] 100 to 200 per second Three identical equations

Table 2-2 (Continued)

Function	Descriptor	Comments
3. Direction Cosines	Formula evaluation (Incremental)	1000 to 10,000 per second Nine identical equations
4. Acceleration Trans- formation	Matrix Multiplication (Incremental)	100 to 200
5. Navigation Computation	Formula evaluation (Incremental)	100 to 200 per second Two non-identical equations
6. Output computation	Coordinate Trans- formation	1 to 10 per second
7. Error Signal Compu- tation	Sum of Products	100 per second
8. Stability Compensation	8th order difference eq.	100 per second

CONCLUSIONS

The use of computation descriptors to describe the requirements was only partially successful. It can be seen that a large portion of the computation had to be described in terms of more basic descriptors than intended. However, some general conclusions can be drawn:

- 1) The most frequently used descriptor is the sum of products computation. This is particularly true when realizing that some of the other descriptors, such as matrix multiplication and polynomial evaluation, also involve a sum of products computation. The summation operation is somewhat difficult for conventional associative memories and should be given some attention.
- 2) The data handling function, particularly that of data acquisition, appears to have requirements which naturally fit the capabilities of an associative memory. An associative controller would provide the desired adaptive sampling capability as well as flexibility in scheduling and controlling the sensors.
- 3) Data compression is probably one of the most pressing tasks of the on-board computer system. While compression of pictures would provide the most significant gain, compression techniques can be applied to other data sets as well. In view of the importance of this task, it might be feasible to consider the availability of a library of encoding and filtering methods. For each data set under consideration a number of currently preferred combinations of filters and encoders could be applied. The separate results could then be evaluated on the basis of the number of bits needed to represent the data set. The list of preferred encoders and filters might then be modified to reflect the successes and failures.

- 4) The incremental computation required for navigation and control of the lander vehicle represents a type of arithmetic to which associative memories are applicable. It is also significant that this computation is required for a relatively short period; thus it is desirable that this task be performed by something other than special purpose hardware.
- 5) In general the desirability of adaptivity in the computing facility is evident. The computer should have freedom to alter its own processing schedule. This would include saving data for later processing, changing priority of the various experiments to accommodate the relative information rates, and modifying the various processing techniques according to the work load.
- 6) The reliability requirements on the computing facility of a space vehicle are severe, particularly if the facility is in any way centralized.

SECTION III

A SURVEY OF ASSOCIATIVE MEMORY ORGANIZATIONAL APPROACHES

INTRODUCTION

The associative memory concept was first described by Slade¹ in 1956. Since then many papers have been written on the subject with terms such as content addressable memory and search memory being used almost as frequently as associative memory. Also, varying definitions have been used in describing such memories. Therefore an appropriate starting place for this survey of associative memory organizational approaches is in the area of definitions.

First an associative memory is defined as a device having the capabilities of storing binary words and of carrying out a set of operations on these words. A significant part of the definition is that the operations are performed simultaneously over all words.

The term "cell" will be used to refer to the logic-storage device which has as a minimum the capability of storing a single bit of information. It may include, however, a logic performing capability that can vary from the EXCLUSIVE-OR operation to that of a full adder. Thus an n word, m bit per word, associative memory will contain an $m \times n$ array of cells.

Figure 3-1 indicates the basic parts of an associative memory and the information flow during search operations. In general terms, the search word is applied through a mask register to the memory array where a comparison is performed simultaneously with each stored word. The results of the comparison are transmitted on a per word basis to the Results Register. Masking of selected bits of the search word is handled through use of a mask register. Words can also be masked out of a search by either assuring that a mismatch will occur or by blocking the output from the masked word.

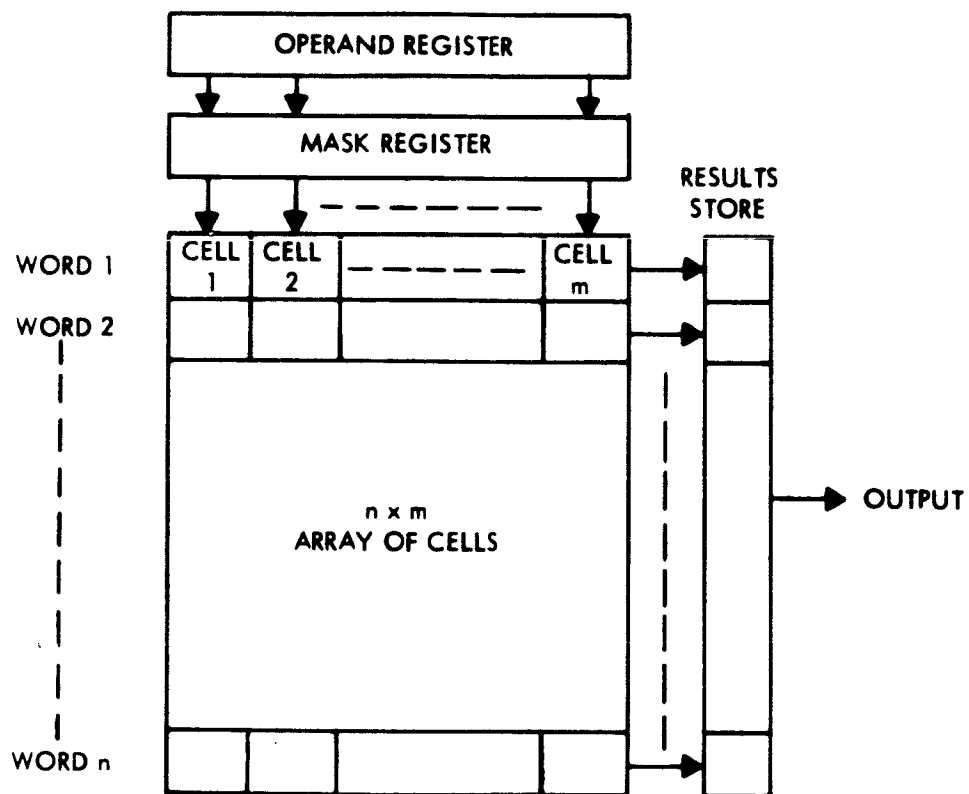


Figure 3-1. Functional Block Diagram

The terms bit slice and word slice are used to describe the read, write, and processing modes of operation. Referring to the block diagram (Figure 3-1), a word slice is simply a single m-bit word, while a bit slice is a slice one bit wide in the vertical direction. The i^{th} bit slice therefore would contain the i^{th} bit of each word in the memory.

The operations performed by an associative memory can be divided into search operations and processing operations. The difference is that processing operations in some way change the contents of the memory, while the search operations do not.

A number of the basic search operations normally performed in an associative memory are defined below. Searches performed as the intersection or union of basic searches are also described. To simplify the description of the operations, all numbers are assumed to be positive. Negative numbers can be handled in most cases with little difficulty.

The algorithms for these searches are normally quite dependent on the organizational approach and are therefore described in a later section. In each case, performance of the search operation will identify a set of words. The next step is to either readout the identified set, generate the address of the identified set, or generate a count of the number of words in the set.

BASIC SEARCHES

Equality Search

The equality search is the operation of finding all stored words which are equal to the search word in all unmasked positions. Logically this operation can be defined as follows where X is a stored word and S is the search word:

$X = S$ if and only if the following Boolean equation is satisfied

$$\bigvee_{i=1}^m (X_i \bar{S}_i + \bar{X}_i S_i) = 0$$

or alternatively the equivalent equation

$$\bigwedge_{i=1}^m (X_i S_i + \bar{X}_i \bar{S}_i) = 1$$

is satisfied where $\bigvee_{i=1}^m$ and $\bigwedge_{i=1}^m$ denote the logical sum and logical product

of m terms, respectively.

Inequality Search

The inequality search is that operation of finding those stored words X which are greater-than or of finding those which are less-than the search word S . The inequality search can be expressed logically as follows:

$$X > S \text{ if and only if } X_d = 1 \text{ and } S_d = 0$$

where the d^{th} bit is the most significant bit in which $X_i \neq S_i$.

$$X < S \text{ if and only if } X_d = 0 \text{ and } S_d = 1$$

where the d^{th} bit is defined as above.

Maximum (minimum) Search

This search operation is that of finding the stored word having the largest (smallest) magnitude.

Proximity Search

The proximity search is defined as the operation of finding the stored word that comes closest to matching the search word in terms of the number of matching bits. It is possible that such a search could be mechanized as a basic operation; however in all the approaches to be described the proximity search is performed by doing an equality search on each bit, count the number of mismatches, and perform a minimum search on the contents of the counters.

Intersection of Searches

The intersection of two or more sets of words, each of which satisfies a basic search, is simply that set of words common to all the basic sets. This is equivalent to a logical AND of basic searches. The most straightforward way of performing the intersection of searches is to use the results of the first search as the input set to the next search. Assuming that the intersection of searches is performed in this manner, the composition of searches, in which the order of performing the searches is important, will be included in this category.

Two familiar searches which are normally performed as an intersection of searches are described below.

Between Limits Search

This search can be performed as the intersection of two inequality searches. Using a "lower limit" as the search word, a greater-than search is performed first to identify the set of words greater than the lower limit. Then with the upper limit serving as the search word, a less-than search is performed to identify the set of words less than the upper limit. The intersection of these two sets is the desired output set.

Next Larger (smaller) Search

This operation is to find that word in memory which is closest in magnitude to the search word with the additional stipulation that it be greater-than (less-than) the search word. This operation is performed by first using the inequality search to find all words greater-than (less-than) the search word and then, using that set as the input set, performing a minimum (maximum) search to find the next larger (smaller) word.

Union of Searches

The union is that set of words satisfying one or more of a number of searches. This is equivalent to a logic OR of basic searches.

PROCESSING OPERATIONS

A number of processing operations are now described. Recall that the purpose of these operations is to alter all or part of the contents of the memory.

Addition ($X_1 + S$, $X_2 + S$, - - - - - , $X_n + S$)

This operation results in the addition of a quantity S located in the operand register to a set of stored quantities X , where each X is stored in a different word of the memory. The result either replaces the stored quantity or is stored in another field of each word involved in the operation.

Field Addition ($X_1 + Y_1, X_2 + Y_2, \dots, X_n + Y_n$)

This type of addition involves two quantities stored in the same word of the memory. The quantity X is to be added to the quantity Y in each word and the result is to be written back to replace either X or Y or to be stored in a third field.

Summation ($X_1 + X_2 + X_3 + \dots + X_n$)

In this case it is desired to produce the summation of a set of quantities X stored in separate words of the memory. The result may either be written into one of the words or placed in an external register.

Counting ($X_1 + 1, X_2 + 1, X_3 + 1, \dots, X_n + 1$)

The desired capability is that of simultaneously incrementing (or decrementing) a set of quantities stored in separate words of the memory. It can be seen that this is actually a special case of the first type of add operation.

Shifting

This is simply the operation of simultaneously shifting a selected set of quantities which are located in the same field of different words.

Complementing ($X_1 \rightarrow \bar{X}_1, X_2 \rightarrow \bar{X}_2, \dots, X_n \rightarrow \bar{X}_n$)

This operation is that of simultaneously replacing each quantity X by its complement \bar{X} .

$$\text{Logical Sum} - (X_1 \cup S, X_2 \cup S, \dots, X_n \cup S)$$

The logical sum is a bit-by-bit OR'ing of the contents S of the operand register with the stored quantities X. The result is stored either in a second field of each word or in place of X.

$$\text{Logical Product} (X_1 \cap S, X_2 \cap S, \dots, X_n \cap S)$$

The logical product operation is a bit-by-bit AND of the contents S of the operand register and the stored quantities X. The result is stored either in a second field of each word or in place of X.

Floating Point Add

Floating point operations can also be performed in an associative memory by using a combination search and processing operation. Since the algorithm used is the same regardless of the organizational approach, a portion of it is described here.

Normalizing a floating point quantity is one of the more difficult and time consuming tasks required in floating point arithmetic. Assume a floating point number made up of a 15 bit fraction F and an exponent C. Each word also has a scratch pad portion which in this operation is used to store the "shift count".

Cycle 1

Search for 8 leading zeros in F. In all words satisfying the search, shift F eight positions to the left and write a 1 in the fourth bit position (2^3) of the shift counter.

Cycle 2

Search for 4 leading zeros in F. In those words satisfying the search, shift F four positions to the left and write a 1 in the third bit position of the shift counter.

Cycle 3

Search for two leading zeros in F. In all words satisfying the search, shift F two positions to the left and write a 1 in the second bit position of the shift counter.

Cycle 4

Search for one leading zero in F. In all words satisfying the search, shift F one position to the left and write a 1 in the least significant position of the shift counter.

Cycle 5

Subtract the content of the shift counter from the exponent C.

An associative memory which is capable of performing one or more of the processing operations described above as well as the basic search operations is commonly called an associative processor.

In this survey the basic method of categorizing associative memories is by organizational approach. This is felt to be better than to survey by device since the capability limitations of an associative memory depend primarily on the method of organization.

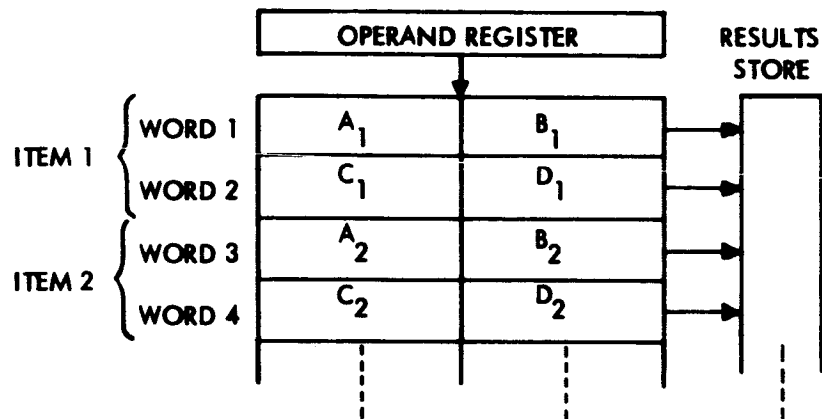
The major differences in the organizational approaches to be described are concerned with the capabilities of the cell, the writing capabilities, and the external facilities provided on a per word basis. There are, however, a number of variations, particularly in terms of the external or peripheral facilities which have an identical effect on all organizational approaches to be described. These variations and their effects on the processing capabilities of the associative memory will be described in the next section prior to the description of organizational approaches.

FACILITIES FOR STORAGE AND PROCESSING OF RESULTS

The Results Store (Figure 3-1) has been identified as the place where the results of a search or arithmetic operation are stored and interpreted. The facilities included in the Results Store can vary in many ways - the storage capability may vary from a single bit to multiple bits of storage per word; a shift (vertical in Figure 3-1) capability may or may not be included; and special logic to determine such things as the number of 1's in a register may be included. The effects of some of these trade-offs are highly dependent on the organizational approach. Examples of these are the number of bits of external storage per word, and the comparison or processing logic associated with each word. There are several trade-offs to be made in this area, however, in which the effect on the capabilities is the same regardless of the organizational approach. Thus these variations can be discussed independent of the various organizational approaches to be described later.

Shift Capability

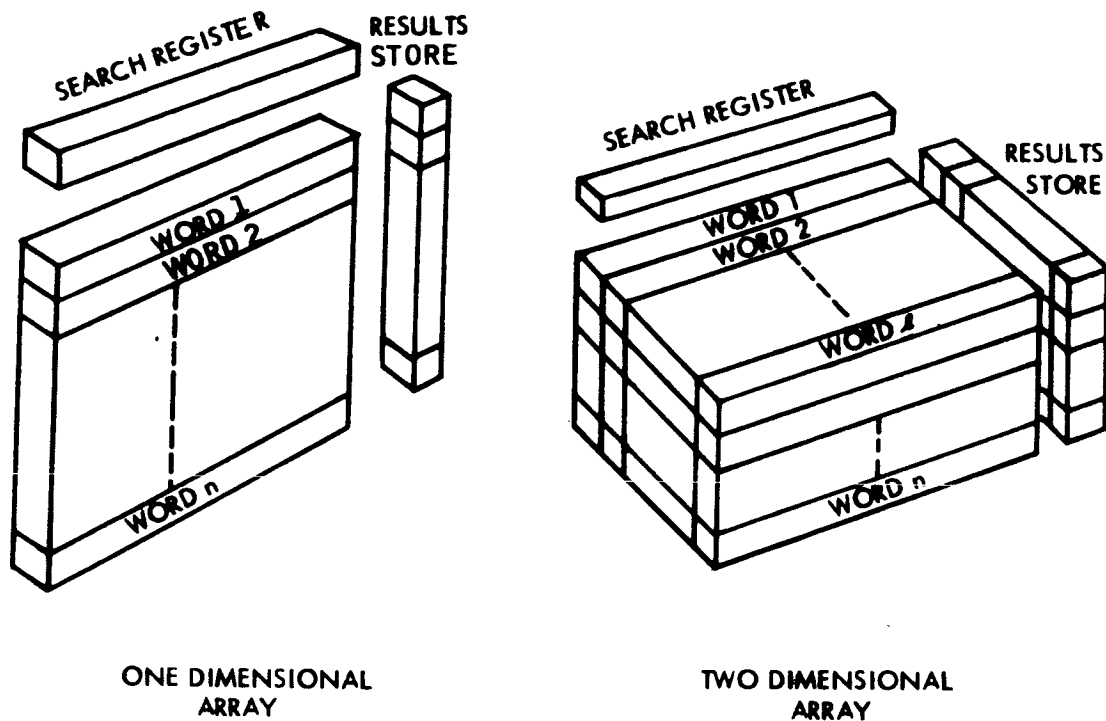
If one of the registers in the Results Store has a shift capability (vertical in Figure 3-1), interword searches can be performed. To illustrate the need for such capabilities consider the situation where two words of the associative memory are required to store the four fields A, B, C, and D of an item.



Assume there is a requirement to perform an equality search on fields A and D. This must be done by first searching over the first field in all odd numbered words and then shifting the results down one in the results store. The second half of the search, which is over the second field, is restricted to those even numbered words paired with odd numbered words satisfying the first search.

It can be seen that the capability of shifting either up or down will effectively give each word two nearest neighbors. Additional shift capabilities or interconnections of register stages can be provided to give each word four nearest neighbors. The array of words in this case can be pictured as a two dimensional array which will naturally fit some problems better than the one dimensional organization.

The usual one dimensional array and the two dimensional array are illustrated on the following page:



It should be kept in mind that there is no difference in the connectivity of the words themselves but only in the Results Store.

Logic To Determine if at Least One Match

In order to carry out the maximum (minimum) search it is necessary to know after the i^{th} bit slice comparison (with a 1 (0) in the search register) whether any of the words still being considered for maximum (minimum) value has satisfied the search. This is needed since a decision must be made regarding the set of words to be included in the $i+1^{\text{th}}$ bit slice comparison. If one or more words satisfied the i^{th} bit slice comparison then that set should be the input set for the $i+1^{\text{th}}$ bit slice comparison. On the

other hand if no words satisfied the i^{th} comparison, the input set used for the i^{th} comparison should be retained for the $i+1^{\text{th}}$ comparison. Thus the capability of determining if at least one match has occurred is essential for the maximum (minimum) search.

Logic to Determine if Exactly One Match

The facility to determine if exactly one match exists is also useful for the maximum (minimum) search. As the search proceeds from the most significant end, if on the i^{th} bit slice comparison only one word is found to match the "1" (0) in the search register, then that word is the maximum (minimum) value and the search can be stopped. This then will in some cases allow a search to be terminated early, whereas without this facility the search must continue through all bit slices.

Logic to Determine Exact Number of Matches

The facility for determining the exact number of matches resulting from a search operation is useful for one of the processing operation described in the previous section. The operation is addition where a summation of many numbers in the associative memory is required². This can be done by searching for 1's in the least significant bit position of all words.

The number of 1's is then added into an external parallel accumulator. The next step is to search for 1's in the second bit position of all words. Again the number of 1's is determined and is added into the accumulator in a position one place to the left of the previous step. This procedure is continued through the most significant bit of the set, at which time the summation of all the numbers is contained in the accumulator.

There are a number of applications which can utilize such a facility. One example is library-type information retrieval where it may be desired

to insert keywords until such a time that the number of items satisfying the search is below some particular value. Another problem area where such a facility is useful is in statistical analysis.

Word Select Ladder

A word select ladder is that logic which allows rapid sequential selection of 1's in the Results Store and thus words satisfying a search. This sequential selection is necessary if the words are to be read out in a word slice mode or if it is desired to generate an address for each word. In the case where an address is desired, a two dimensional arrangement of the Results Store will speed the process³. In this case the Results Store is arranged as a rectangular array. The address of a 1 in the array is generated by identifying the column and the row to which it belongs.

ORGANIZATION APPROACHES

The organizational approaches described below have been selected as being representative of a larger number of possible approaches. It is assumed that in each case, all the peripheral facilities described in the previous section are included unless it is specifically pointed out that they are excluded. Each of the basic approaches is allowed to have variations that effect either the list of devices applicable to the approach or its capabilities to perform one or more of the processing operations. One such variation which has an effect on the device selection is common to all approaches and is discussed here. This is concerned with the reading operations. All the approaches described are assumed to have both bit slice and word slice reading capabilities. Bit slice reading in most approaches can be obtained as a special case of a bit slice comparison. Word slice reading, however, may or may not be included. The absence of word slice reading does not directly effect the processing capabilities, but it does limit the overall effectiveness of an associative memory on some applications. Devices that are otherwise suitable but lack the capability of word slice reading are identified in the discussion of mechanization considerations for each approach.

Approach 1 - Minimum Associative Memory

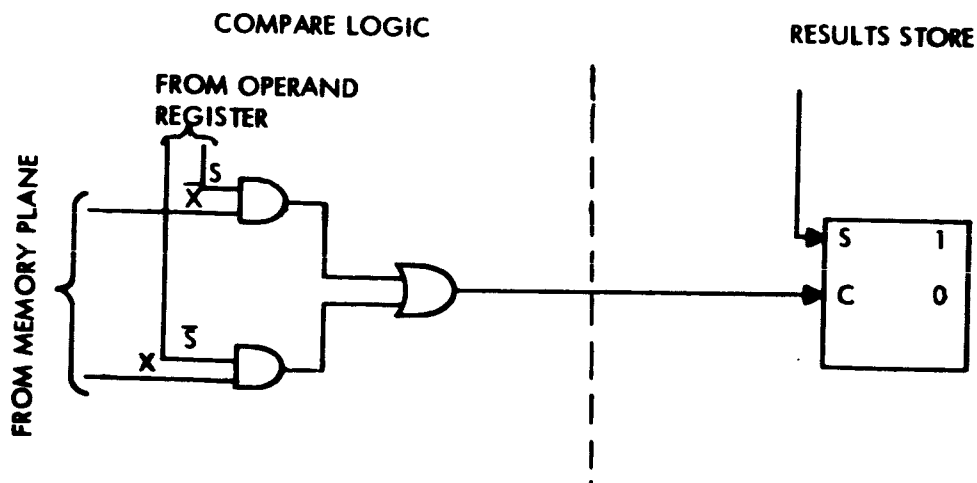
Description

This approach is considered the minimum associative memory because it requires no local logic. All operations are carried out in a bit slice mode with logic completely external to the memory plane. Thus the memory plane is simply an array of memory elements and may be very much like

a conventional random access memory depending on the peripheral hardware provided. A block diagram of the Minimum Associative Memory is shown in Figure 3-2. It can be noted that random access to a bit slice is provided by decoding a bit address supplied by a controlling unit. Likewise a word address decoder is provided to select a word for reading or writing through the input-output register. An operand register and a mask register are provided with their contents shifted serially to the External Logic in synchronism with the bit slice readout of the memory plane for any of the search operations. The temporary results which must be related from one bit slice to the next and the final result of all search operations are stored in the Results Store. In this approach, the final result will be converted to an address for use by the controlling unit.

Capabilities

Equality Search -- The equality search is performed by providing the contents of all stored words a bit slice at a time to the Compare Logic in synchronism with the search word stored in the Operand Register. The Compare Logic associated with each word must perform the EXCLUSIVE-OR logic function between each stored bit and the corresponding bit of the search word. This logic might take on a form as shown below:



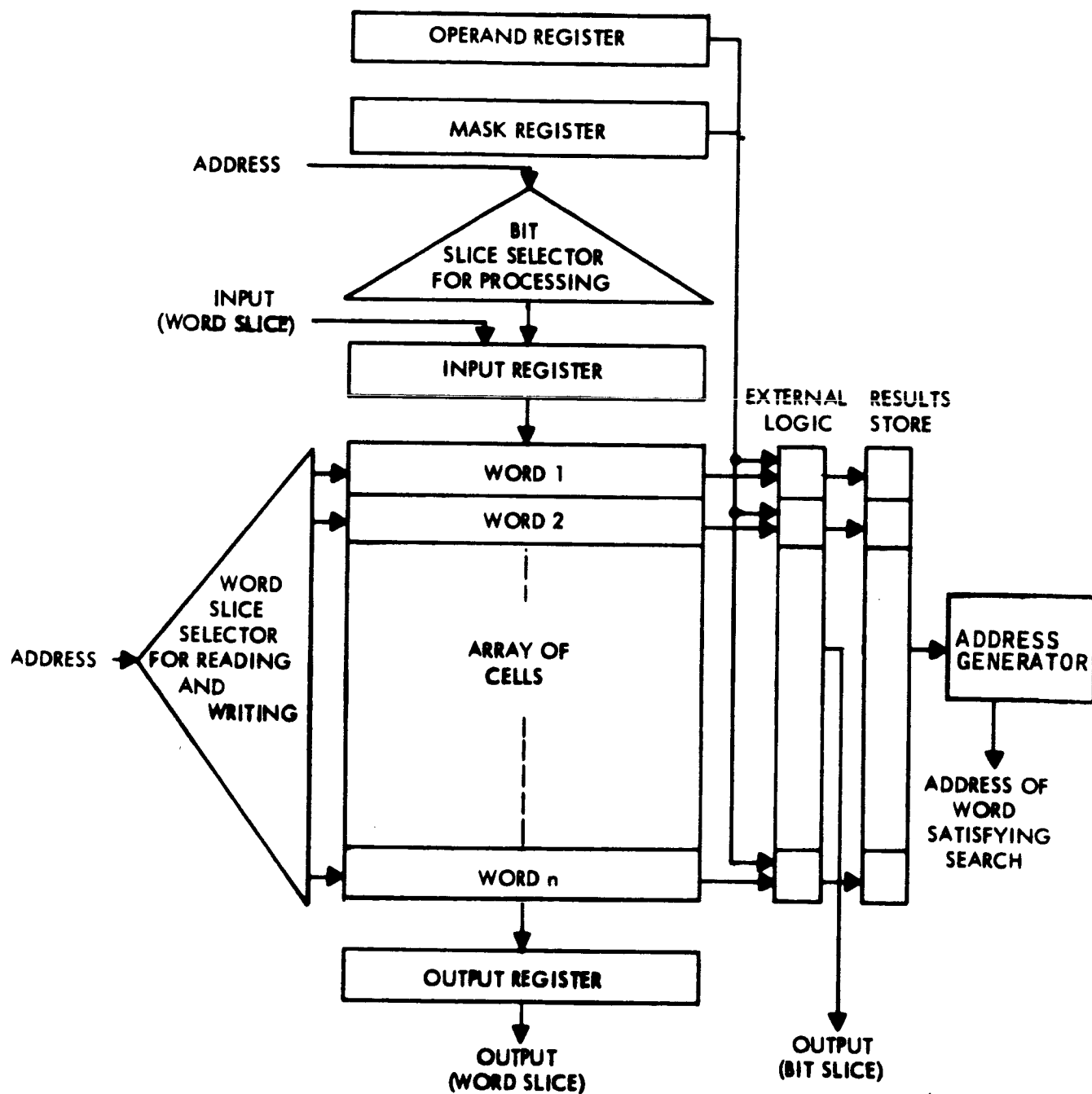


Figure 3-2. Approach 1 - Minimum Associative Memory

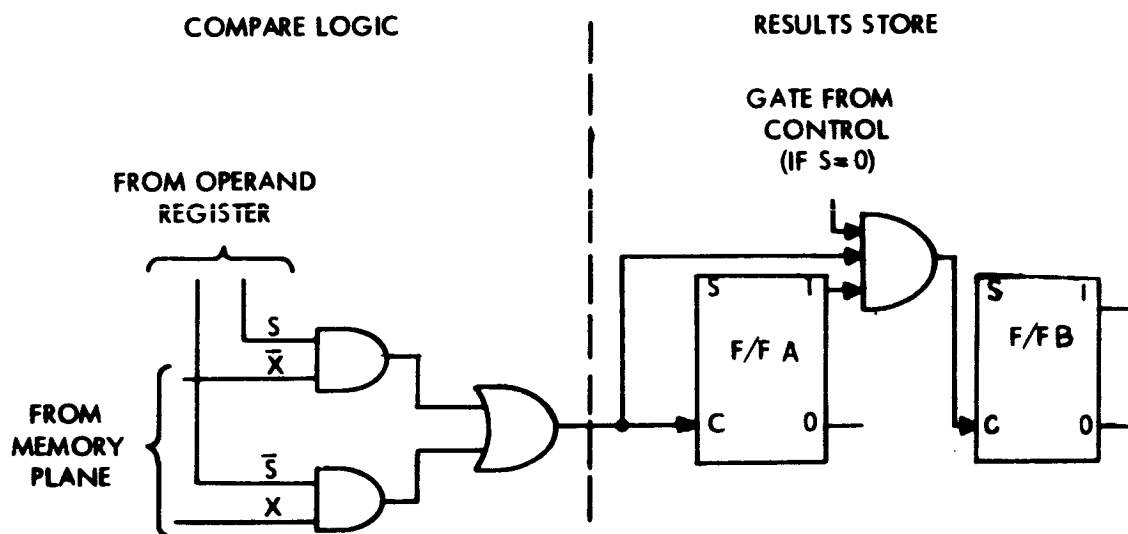
Assume that initially all F/F's are set. Any mismatch between the search word S and the stored word will cause the F/F to be cleared. Thus when the search is completed, a F/F containing a 1 will correspond to a word equal to the search word.

The time required to perform the equality search is equal to the number of bits involved in the search times the time required to read a single bit slice from the memory. The assumption is made that the operation is limited by the speed of the memory plane rather than the Compare Logic.

Inequality Search -- In this memory the inequality search must be performed in the bit slice mode such as was described for the equality search. There are two algorithms which can be employed -- one starts from the least significant end of the word, and the other starts from the most significant end. The algorithm starting with the most significant end is probably the best choice, however, since the maximum (minimum) search (described next) must proceed from the most significant end. Thus this algorithm is described below. The algorithm starting from the least significant end is described in Approach 1a, which is a variation of this approach.

The general description of this algorithm is taken from Reference 4. Examine the most significant bit of the search word. If it is a 1, then all words mismatching in that position are smaller and all those matching are indeterminate. If the most significant bit is a 0, all words mismatching are larger and those matching are indeterminate. Repeat the above procedure for successively less significant bits on those words that are still indeterminate. When the operation is completed, the stored words will have been separated into three sets - those larger than the search word, those smaller than the search word, and those equal to the search word.

It is important to point out that no additional logic is required on a per word basis to carry out this operation. There is simply the requirement for a decision to be made on the basis of the contents of the Results Store and the contents of the current position of the search word. However an additional bit of storage is required to store the results on those words on which a decision has been made. The logic and storage might be organized as shown below:



Assume that both A and B flip flops (F/F) are initially set. As in the Equality search, any mismatch signal is used to clear the A F/F. The B F/F, on the other hand, can be reset only by the mismatch condition of $S = 0$, $X = 1$. Once a mismatch has occurred, the mismatch signals from subsequent bits have no effect on either the A or B F/F's. When the operation is completed the state of the two flip flops are interpreted as follows:

<u>AB</u>	<u>Interpretation</u>
11	Equality, $X = S$
10	Will not occur
01	Inequality, $X < S$
00	Inequality, $X > S$

The time required to perform the inequality search in this approach is equal to the number of bits involved in the search times the time required to read a signal bit slice from the memory. Note that this is the same as the time required to do the equality search since it can generally be assumed that the limiting factor is the readout of the memory array rather than the external operations.

Maximum (minimum) Search -- The maximum (minimum) search is the operation that consists of finding the stored word having the largest (smallest) magnitude. This search must proceed in a bit slice manner and must start at the most significant end. It also has the somewhat unique characteristic that there is a dependency between words. The results of each bit slice comparison must be interpreted before the next step can proceed.

The operation is initiated by placing 1's (0's) in all positions of the operand register and comparing in the most significant bit position. The following action is required:

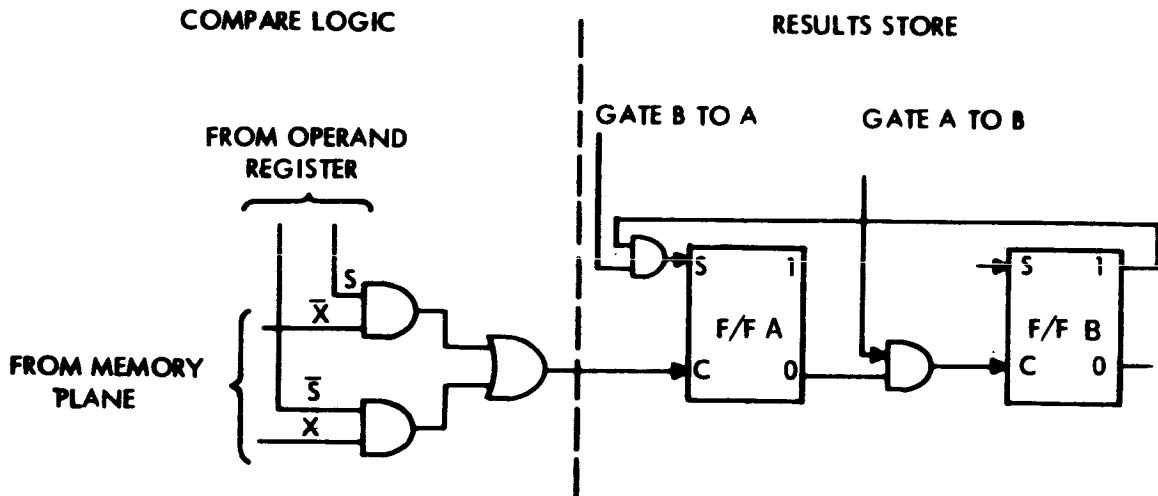
- a) If a match condition exists for only one word, that word is the maximum and the operation is terminated.
- b) If a match condition exists for more than one word, all words receiving mismatch signals are deleted from further consideration and the operation is repeated for the next bit.
- c) If no match condition exists, all words are retained and the operation is repeated for the next bit.

The operation is ended when the match condition exists for a single word or when all bits have been compared. In the latter case more than one word may contain the maximum value.

It can be recognized that there is a requirement for two bits of external storage per word for this operation. The set of words still under

consideration for maximum value at the start of each bit slice comparison must be stored in two sets of F/F's. This is necessary in case the current bit slice comparison results in no matches.

The logic and storage requirements are satisfied by the arrangement shown below:



Prior to each bit slice comparison both the A and B F/F's contain 1's if the corresponding word is still in consideration for maximum value. If the i^{th} bit slice comparison produces a mismatch the A F/F will be cleared. The next step depends on the number of 1's in the A register (all A F/F's). If no 1's exist, the "Gate B to A" signal is energized. If at least one 1 exists in the A register, the "Gate A to B" signal is energized. In either case the A and B F/F's will be the same prior to the comparison of the next bit slice.

The time required to perform a maximum (minimum) search is somewhat longer than that needed to do the equality and inequality searches. This is because the external processing and decision making operations are assumed to be more time consuming than the bit slice readout operations.

Proximity Search -- The proximity search cannot be performed in this associative memory.

Intersection of Searches -- The intersection of searches can be conveniently performed in this memory by simply using the results of one search as the input set to the second.

Union of Searches -- Only a restricted type of union of searches can be handled. The only way it can be done is to transfer the results of the first search to the second F/F associated with each word, perform the second search using only the first F/F, and then logically OR the contents of the two F/F's. The restriction is that only the first search of a sequence of searches is allowed to use both F/F's.

Processing Operations -- No processing operations are possible in this organizational approach since there is no capability for writing a bit slice.

Mechanization Considerations

Requirements on the Device -- The fact that the cell in this associative memory contains no logic makes it possible to mechanize it with a wide variety of memory devices. The memory device must be capable of a non-destructive readout (NDRO) mode, however, since no bit write capability is provided for re-writing after the bit slice has been read out for comparison purposes.

It can be noted in Figure 3-2 that reading of the memory device is required in two directions so that both a bit slice and a word slice can be read from the memory. Writing, however, occurs only by word slice.

Applicable Devices -- Many devices are capable of satisfying the requirements of this approach since these requirements are not very severe. Some of these devices, however, have capabilities much greater than required and would therefore not be competitive with the others in terms of size, power consumption, or cost. Those that appear to match the requirements most closely might be broadly classified as non-destructive readout

magnetic devices although not all of these are readily applicable. Non-destructive readout devices are required because the approach does not specify a bit slice write capability and therefore anything that is read out in the normal bit slice mode is lost if the readout is destructive. In this approach the memory elements operate in the same manner as a conventional coincident current or word arranged memory except for the additional requirements of the interrogate circuits, sense lines, and sense amplifier required to read out a bit slice.

Multi-Aperature Devices -- Biax cores, transfluxors, and other multi-perature devices provide a relatively simple NDRO mode of operation and writing in memories employing these devices can be accomplished in much the same way as in an ordinary core memory. The Biax element has the important advantage of a very high NDRO readout speed which is usually limited more by the electronics than by the element itself. Thus the search speed of a Biax memory (100 nanoseconds per bit or less) would be much greater than that of a memory employing any of the other multi-perature devices. Transfluxors and other non-orthogonal multi-perature devices, on the other hand, have the advantage of somewhat lower costs and are more readily adaptable to thin film batch-fabrication schemes.

Magnetic Cores -- Conventional ferrite cores can also be operated in an NDRO mode. Two examples of memories employing this type of operation are the flux-lok memory and those employing minor loop readout. These methods of operation however impose stricter tolerances on the operating parameters than those required by the conventional DRO mode and are therefore not commonly employed. The maximum interrogation rate for a ferrite core memory operated in the NDRO mode is lower than that of a Biax memory, but the cost is also much lower, especially for large memories. It would also be extremely difficult to obtain the necessary bi-directional readout capability with these NDRO readout methods.

Thin Magnetic Films -- Thin magnetic films of both the flat and cylindrical types can be used in this type of memory. They offer potential advantages of high speed, small size, reduced power consumption, and low cost. Problem areas such as uniformity, creep, and aging still exist but they are apparently being solved. Two-dimensional readout has also been a problem, particularly in the cylindrical configuration, but at least one solution to this problem has been proposed. The open flux structure of the flat thin film results in a large de-magnetizing field. This imposes a maximum value on the thickness to area ratio of the film. As a result, a compromise must be made which limits the packing density and maximum sense line output voltage. The cylindrical thin films, the most common of which are the plated wire elements, are not subject to these problems because the circumferential easy direction results in a closed-flux structure. Most of the future NDRO thin film memories are expected to employ cylindrical thin films for this reason.

Tunnel Diodes -- The use of tunnel diodes in memories of this type has been suggested. Because of their extremely high switching speeds, tunnel diodes offer potential speed advantages in memory as well as logic applications. In this approach however, a serial search algorithm is employed. As a result the search speed is dependent not only on a switching speed of the storage element but also on the speed of the auxiliary electronics. When a very high speed switching element such as the tunnel diode is used in a memory of this type the external electronics may have the greatest effect on the switching speed. As a result tunnel diode memory may not be faster than a thin film memory when this approach is employed. Tunnel diodes have several disadvantages over magnetic elements. They consume more power, particularly during standby operation; are volatile require addition circuit elements for read-in and read-out; and are probably much less reliable because of the large number of elements and interconnections required at each bit location. For these reasons, except for very small memories, tunnel

diodes should more properly be considered in connection with associate memories employing local logic and cell intercommunication.

Ferroelectrics -- In applications where memory speed is less important than power consumption ferroelectric and ferroelectric elements could be considered. These elements share with the magnetic elements the advantages of NDRO readout, non-volatility, and simplified reading and writing. Maximum readout speeds however are probably in the one to ten microsecond region. They also appear to be adaptable to mass fabrication techniques although considerable research and development may be required before this goal can be obtained.

The transpolarizer, a ferroelectric device similar in operation to the transfluxor, could be used in this application. It consists of two or more ferroelectric elements interconnected to form each storage cell. In the present case, a total of three per cell are required and are connected as indicated in Figure 3-3. This connection provides word slice write capability along with NDRO bit slice and word slice read capability. The lower element in each cell holds the information and the two upper elements provide the non-destructive readout, one for each of the two readout directions. This circuit has not been tested but it appears to be a reasonable solution to the two direction readout problem.

The basic circuit of the NDRO memory cell is shown in Figure 3-4. The polarization of each ferromagnetic element is indicated for each part of the write and read cycles. Note that if a "0" is stored, the read pulses does not affect the polarization of either element; whereas if a 1 is stored, both elements switch and the second writes the 1 back into the cell. The presence of a stored 1 is sensed by means of a resistor that measures the driver current during the switching of an element.

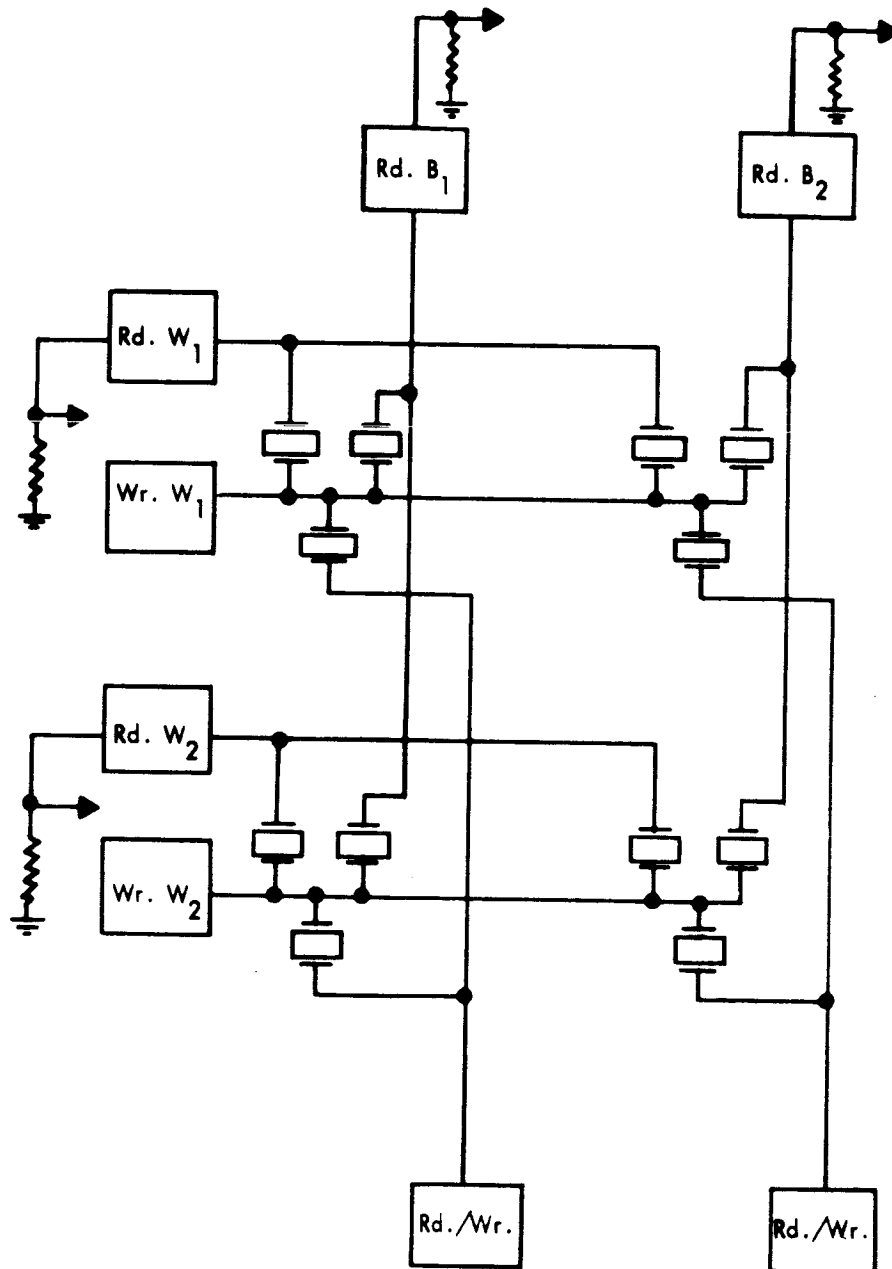


Figure 3-3. Transpolarized Memory With Two Dimensional Readout

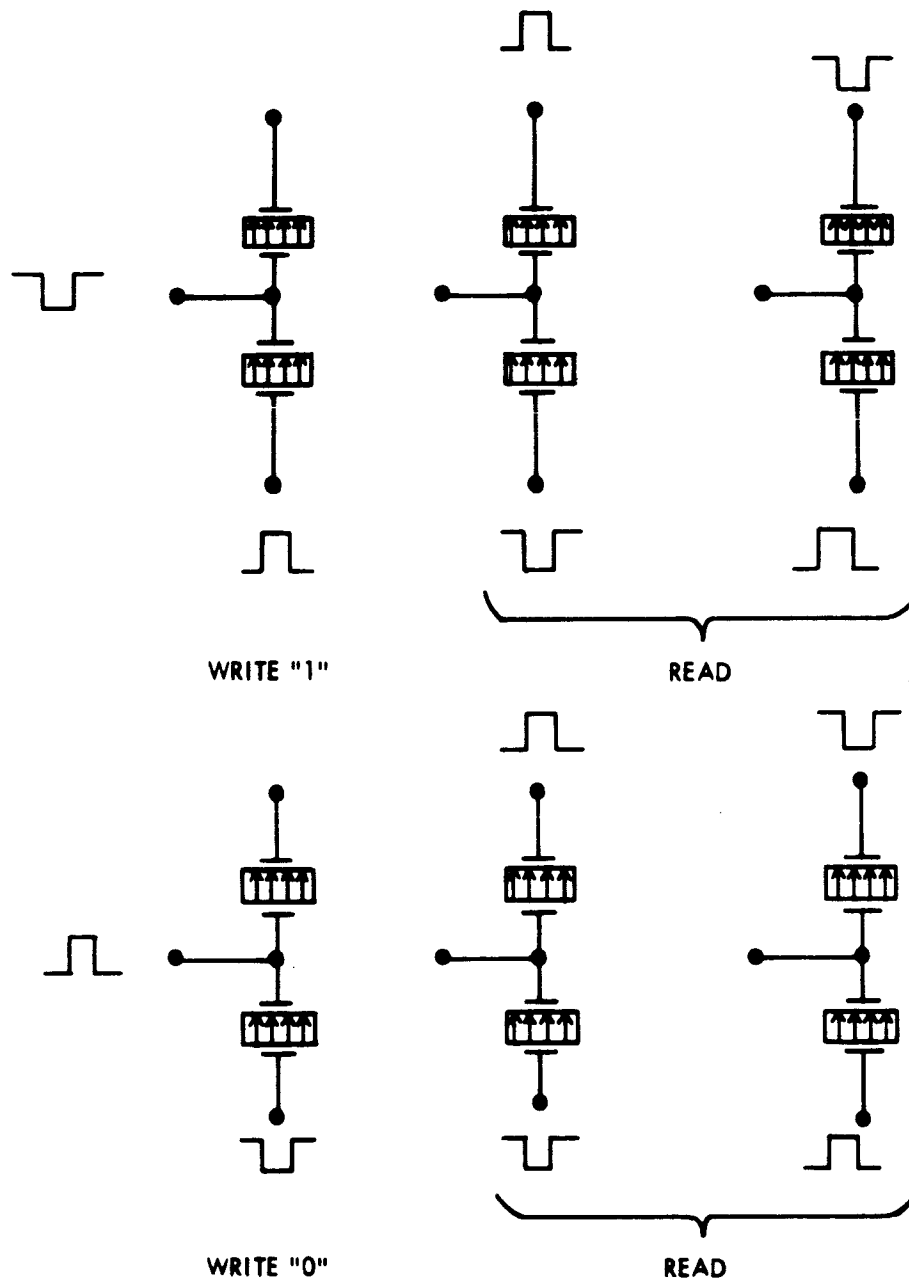


Figure 3-4. Transpolarizer NDRO Readout

Variations of Approach 1

There are a number of variations of Approach 1 which are obtained by deleting one or more of the basic capabilities described previously. In all cases these variations either make it possible to realize the memory with a device which could not satisfy all the previous requirements, or they make a significant change in the capabilities of the unit.

Approach 1a - Fixed Information Memory Element -- This variation involves the use of a fixed information element. This change has no direct effect on the associative operations that can be carried out; however, it allows the use of a different set of elements. Of course the fact that the memory cannot be written into does severely limit the applications but this will not be considered at this time.

Any member of the class of read only elements described would be suitable for this approach. For small size memories, semi-conductor diode coupling elements would be preferable since simple low powered drive and sense circuits could then be employed. For larger memories inductively coupled elements with printed circuit drive and sense lines appear to be advantageous. They are capable of providing very fast search rates, probably at least as fast as 100 nanoseconds per bit, and at the same time are amenable to low cost, mass production fabrication techniques. Recent improvements in the E-core memory have made it very attractive for medium to large size fixed memory applications. Some memories, such as the slug memory, are much more easily modified than the others. Others read only elements described in Appendix B might be used in special applications.

Approach 1b - External Storage for Single Bit Only -- An associative memory required to carry out only the equality search or the inequality search over all words of the memory could be mechanized with a single bit of external storage per word. That this is possible in the case of the equality search is obvious from the description in the previous section. However in the case of the inequality search, the algorithm described below must be used.

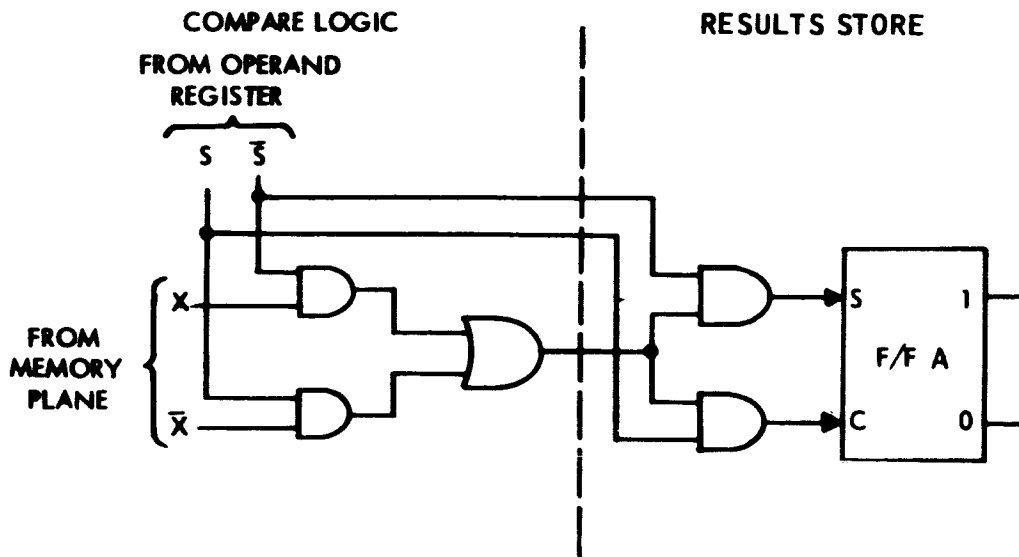
This is also a bit slice algorithm but it starts at the least significant end of the word rather than the most significant end. Also with only a single bit of storage, only a single set of words can be identified rather than three sets (those greater than, those equal-to and those less-than the search word). As in the algorithm described previously the set to be identified by this algorithm can be one of the following four - those words greater-than, those words greater-than-or-equal-to, those words less-than, or these words less-than-or-equal-to the search word.

As an example, consider the steps of the procedure to find those words greater than the search word. Initially the F/F's are set to 0. The comparison starts with the least significant bit and proceeds to the most significant bit. The decisions at the j^{th} word upon comparison of the i^{th} bit are:

- a) If they match do nothing.
- b) If they mismatch and if the i^{th} bit of the search word is 0, set the j^{th} F/F.
- c) If they mismatch and if the i^{th} bit of the search word is 1, reset the j^{th} F/F.

When the processing is completed, all those words greater than the search words will be identified by F/F's in the 1 state.

The logic and storage appropriate for this operation is shown below. Note that the only change required to do a less-than search is to initially set the F/F and when the process is completed all F/F's in the reset condition will correspond to words less than the search word. The greater-than-or-equal-to and the less-than-or-equal-to searches are the same respectively as the two above except the initial state of the F/F is changed.



No other search or processing operations can be performed with this memory.

The devices applicable to this variation are the same as those described for the basic approach.

Approach 2 - Associative Memory with Bit Slice Writing

Description

This approach is like Approach 1 in that it does not make use of local logic. The significant difference between the two is that this memory has the capability of writing a bit slice. This allows the writing-back of results of searches. This is a necessary capability for the performance of processing operations and it also allows the use of the memory plane for storage of results of searches for use on future searches. A block diagram of this organizational approach is shown in Figure 3-5.

The bit slice writing capability is seen to require a driver per word but eliminates the need for address decoding. The information to be written

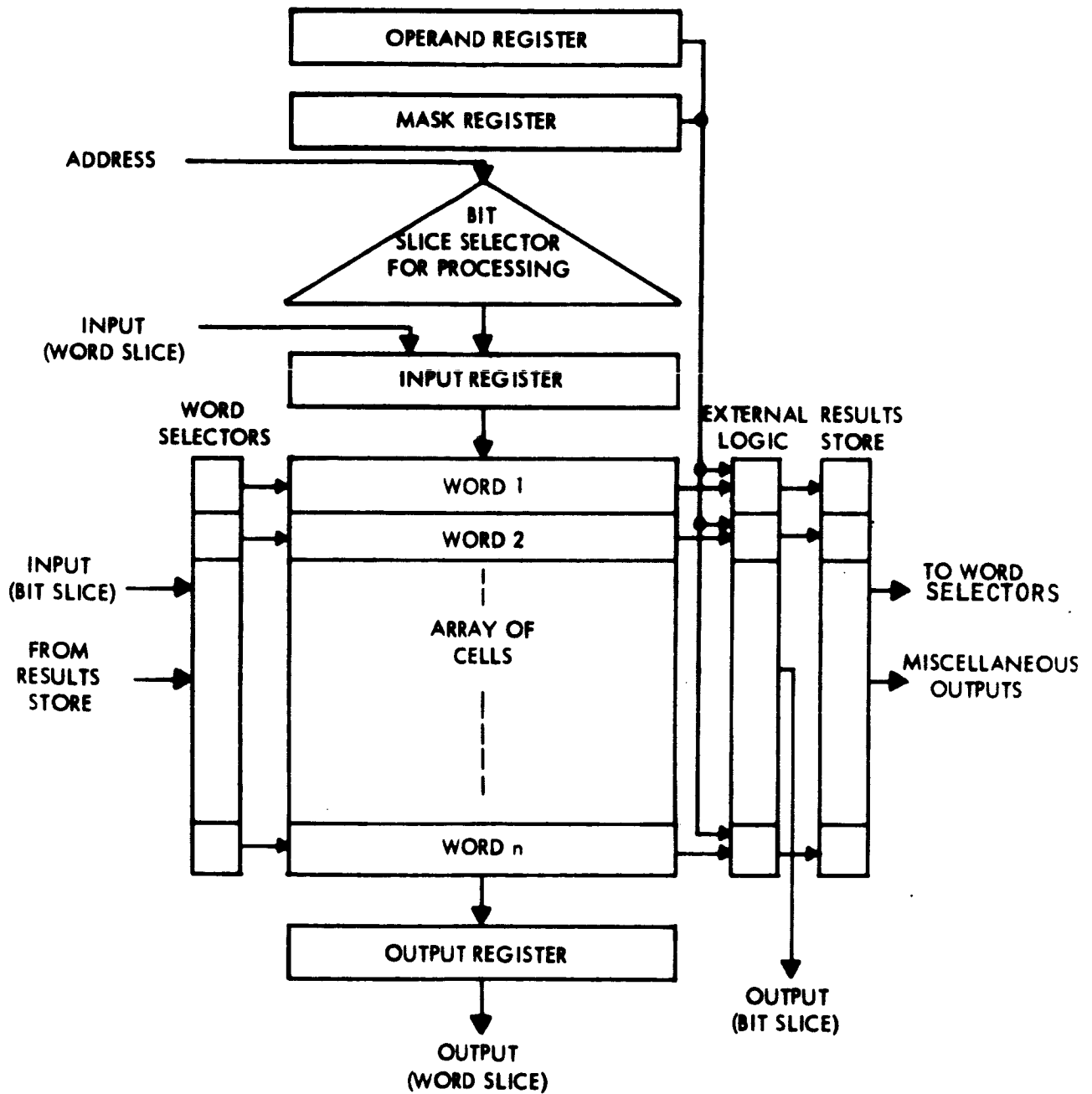


Figure 3-5. Approach 2 - Associative Memory With Bit Slice Writing

is placed in the Results Store, either as the result of a search operation or by gating from the outside, and it is written into a selected bit slice by supplying a path directly from each word-position of the Results Store to each word driver. Writing of a word slice is accomplished by first performing a search to locate a word position and then using the contents of the Results Store to select the word driver for writing. The information to be written in this case is furnished from the Input Register. Of course if the application required writing by address a word address decoder such as was shown in Figure 3-2 could be included.

The reading capability of this approach includes both bit slice and word slice readout. Bit slice reading is accomplished in the same manner as described for Approach 1. Word slice reading is accomplished by performing a search to locate the word, gating the contents of the Results Store to the Word Drivers, and thus transferring the contents of the selected word to the output register.

Other outputs which might be desired from the Results Store, depending on the application, are addresses of words satisfying a search and the number of words satisfying a search.

The access to bit slices for bit slice processing is again accomplished by decoding an address furnished by the Controlling Unit.

The External Logic has the capability of performing the Exclusive-OR logic operation as was the case for Approach 1.

Capabilities

Equality, Inequality, and Maximum (Minimum) Searches -- These searches are performed in exactly the same way as described for Approach 1.

Proximity Search -- The proximity search cannot be performed as a basic operation but it can be performed as a combination of search and processing operations. The procedure is as follows:

- a) Perform a bit slice comparison to determine if a match or mismatch exists in each bit position.
- b) With a portion of each word used as a counter, count the number of mismatches that occur in each word.
- c) When all bits have been compared and counted, a minimum search is performed on the counter to determine which word had the fewest mismatches.

Intersection of Searches -- The intersection of searches is handled by simply using the results of one search as the input set to the next as was done in Approach 1.

Union of Searches -- The union of searches can be handled in this approach without the restrictions imposed in Approach 1. This is done by using a bit slice of memory for the storage of the results of the first search of the sequence while the second is being carried out. The results of the second search are then OR'ed with the first by simply writing them over the results of the first search. This procedure can be continued for any number of searches.

Field Addition ($X_1 + Y_1, X_2 + Y_2, X_3 + Y_3, \dots, X_n + Y_n$) -- This type of add involves the case in which each word of the memory contains an X field and a Y field. It is desired to produce the sum $Z = X + Y$ and store it either in a third field of each word or in place of the X or Y quantity. This operation can be performed by successively re-writing the

bits of X, Y, and the carry bit C into control bit positions. The assignment of bit position in each word of the memory would be as shown below:

WORD 1	X_1	Y_1	CONTROL BITS
WORD 2	X_2	Y_2	CONTROL BITS
		⋮	

A total of seven control bits are required per word to carry out the operation.

Consider the addition operation $X + Y \rightarrow Z$. The Boolean equation for the sum and carry bits are:

$$Z_i = X'_i Y'_i C_i + X'_i Y_i C'_i + X_i Y'_i C'_i + X_i Y_i C_i$$

$$C_{i+1} = C_i X_i + C_i Y_i + X_i Y_i$$

These equations can be written in the form:

$$Z_i = \left[\underset{(1)}{(X_i + Y_i + C'_i)} \underset{(2)}{(X_i + Y'_i + C_i)} \underset{(3)}{(X'_i + Y_i + C_i)} \underset{(4)}{(X'_i + Y'_i + C_i)} \right] ,$$

$$C_{i+1} = \left[\underset{(5)}{(C'_i + X'_i)} \underset{(6)}{(C'_i + Y'_i)} \underset{(7)}{(X'_i + Y'_i)} \right] ,$$

The numbers below each term in the equations above correspond to a control bit in the memory used to generate that term. For instance by storing the quantity X_i , Y_i , and C'_i successively in control bit 1, the term $X_i + Y_i + C'_i$ is generated. Similarly, the other three terms in the Z equation are generated in control bits 2, 3, and 4. The AND function of the four terms of the

Z equation is then generated by doing an equality on control bits 1, 2, 3, and 4 against 1111. This operation is effectively an OR function of the complements; that is, if any of the four is a 0, a 1 will be set in that position of the Results Register. The information in the Results Register at the end of this search is Z_i . The generation of C_{i+1} is carried out in a comparable manner using control bits 5, 6, and 7.

The step by step procedure for $X + Y \rightarrow Z$ is outlined below. At the end of each cycle the carry C is left in the Results Register for use during the next cycle. Thus prior to the first cycle, the Results Register must be cleared to give an initial carry-in of 0.

Procedure for Add Cycle	No. of Reads	No. of Writes
1. Write C_i to control bits 2, 3		1
2. Complement		
3. Write C'_i to control bits 1, 4, 5, 6		1
4. Read Y_i	1	
5. Write Y_i to control bits 1, 3		1
6. Complement		
7. Write Y'_i to control bits 2, 4, 6, 7		1
8. Read X_i and write X_i to control bits 1, 2	1	1
9. Complement		
10. Write X'_i to control bits 3, 4, 5, 7		1
11. Search control bits 1, 2, 3, 4 to generate Z_i	4	
12. Write Z_i (This can be stored in place of X_i or Y_i or in a third field)		1
13. Search control bits 5, 6, 7 to generate C_{i+1}	3	
Totals	9	7

The totals represent the number of reads and writes for a single bit add. Note that it is possible to write the same information into multiple bit slices simultaneously (step 3). This is possible since the information is furnished by the word driver and the bit slice is selected by the bit driver. This will necessitate the capability of energizing more than one bit slice at a time.

Addition ($X_1 + S, X_2 + S, \dots, X_n + S$) -- This operation is performed in this memory in the same manner as the previous type of addition. Instead of reading the Y bits from the memory plane, they are read from the search register. Thus the number of read operations from the memory plane will be one less per cycle than in the previous case.

Counting -- Counting simultaneously in each word of the memory is performed by the following algorithm. A single control bit in each word is used to store the carry C_i and the field X is used to store the counter. It is assumed that an initial carry, which is actually the identity of those counters to be incremented, is located in the Results Store.

- a. Write the contents of the Results Store in control bit 1.
- b. Search for $X_i C_i$. If a match exists write $X_i = 0$.
- c. Search for $\bar{X}_i C_i$. If a match exists write $X_i = 1$ and $C_{i+1} = 0$.
- d. Repeat steps b and c for successively more significant bits of X until all have been processed.

It can be seen that four bit-slice reads (for the two searches) and 3 bit-slice writes are required for each bit of the counter.

Shifting -- Shifting within each word can be performed only by reading and re-writing each bit to be shifted. This requires bit slice reading and writing and thus can be performed simultaneously for all words or for a

selected subset of words. An unusual feature about this method of shifting is that the time required to shift is independent of the amount of the shift and is directly dependent of the number of bits involved in the shift.

The time required to perform the shifting on a single bit regardless of the amount of the shift is 1 read time and 1 write time.

Complement -- The complement operation on a set of words is performed by reading in a bit slice manner, complementing the contents of the Results Store, and rewriting to the same bit slice.

The time required for this operation is 1 read time and 1 write per bit, assuming that the time required to complement the contents of the Results Store is negligible.

Logic sum ($X_1 \cup S, X_2 \cup S, \dots, X_n \cup S$) -- The logic sum of a set of stored quantities X and the contents of the operand register S is performed in this memory by examining each bit of S in any order desired and doing the following:

- a. If the i^{th} bit of $S = 1$, write $Z_i = 1$
- b. If the i^{th} bit of $S = 0$, perform an equality search in the i^{th} bit position
 - 1) If a match signal is received, write $Z_i = 0$
 - 2) If a mismatch signal is received, write $Z_i = 1$

The time required for this algorithm depends on the operand S . If the operation is performed over m bits, the average time is $\frac{m}{2}$ reads and m write.

Logic product ($X_1 \cap S, X_2 \cap S, \dots, X_n \cap S$) -- The logic products of a set of stored quantities X and the contents of the operand register S is performed by examining each bit of S in any order desired and doing the following:

- a. If the i^{th} bit of $S = 0$, write $Z_i = 0$
- b. If the i^{th} bit of $S = 1$, perform an equality search in the i^{th} bit position
 - 1) If a match signal is received, write $Z_i = 1$
 - 2) If a mismatch signal is received, write $Z_i = 0$

The time for this operation is the same as for the logical sum.

Mechanization Considerations

Requirements on the Device -- The requirements on the device to mechanize this approach are quite similar to those of Approach 1. In fact the only difference is the additional requirement to be able to write a bit slice. The inclusion of a bit slice writing capability, however, allows the use of a destructive readout (DRO) memory element. An additional incentive to considering DRO devices is that their write speed is generally faster than that of an NDRO device. If the application is heavy on processing, the write speed is vitally important since the number of writes is nearly as great as the number of reads. However, other consideration such as power still make an NDRO device advantageous.

Applicable Devices -- The requirements of this approach permit all of the elements described under Approach 1 to be used here as well. Because of the additional required bit slice write capability those elements which require large word-write currents may not be acceptable for large memories.

Additional elements which could not be considered in Approach 1 but which can be used in Approach 2 include ferrite and metal cores operated in the DRO mode. These devices offer the advantage of extremely low cost particularly when very large memories are under consideration. The ferrite sheet (or monolithic ferrite) memory could also be used in this approach. Since this type of memory is made by a batch fabrication process it should be less expensive than conventional core memories once production problems have been solved. It is also expected to be somewhat faster and to require less power than the conventional types.

Variations of Approach 2

The capabilities of this approach for arithmetic operations can be improved appreciably by increasing the external logic facilities. The basic approach described above uses a minimum amount of external logic (only the Exclusive-OR function). The logic can be increased in any number of steps until the point is reached where a complete serial adder exists for each word of memory. While it is realized that there are many perfectly reasonable part-way points, only the serial adder per word variation will be described. (Approach 2a).

Two other variations are also described. Approach 2b is simply Approach 2 with only a single bit of external storage per word. Approach 2c is a relatively severe variation; the random access to a bit slice capability is replaced by a cyclic access capability.

Approach 2a - Serial Adder per Word -- In this approach, the External Logic has been expanded to the extent that a complete serial adder is provided for each word. This change, provides increased arithmetic capabilities as compared to Approach 2. With the carry bit stored in the serial adder or in the Results Store a single bit add of any of the types discussed can be performed by the following algorithm:

- a) Read and Store X_i
- b) Read Y_i (either from memory or from the operand register) and generate Z_i and C_{i+1}
- c) Write Z_i

Assuming the memory reading and writing operations are limiting, a single bit add can be done in 2 read times and 1 write time.

The serial adder per word approach has an additional potential advantage. If the requirement for performing arithmetic on a small set of operands (adding two numbers, for instance) occurs, these numbers can be stored in the memory in the bit slice direction and the set of serial adders could be connected to act as a parallel adder as in a conventional computer. In fact this approach used in this manner is not much different than a conventional random access memory along with a conventional parallel arithmetic unit.

The mechanization requirements for this approach are much like those for Approach 2. The only significant difference is that the reduction in the number of write operations relieves to some extent the need for a high speed write.

Approach 2b - External Storage for Single Bit Only -- The effect of restricting external storage to a single bit per word was previously discussed as Approach 1b. It was found to limit rather severely the capabilities of Approach 1. The effect here is somewhat different because Approach 2 has a bit slice writing capability. Therefore a bit slice within the memory plane can serve as the second bit of storage required on a per word basis for operations such as the maximum (minimum) search. It can also serve to store the identity of a selected subset of words if it is desired that not all words be included in a given search.

As an example of how this capability can be used, consider the maximum search. Recall that the set of words under consideration for maximum value prior to each bit slice comparison must be retained until after the comparison in case none of the words satisfy the search. The algorithm would have the following steps:

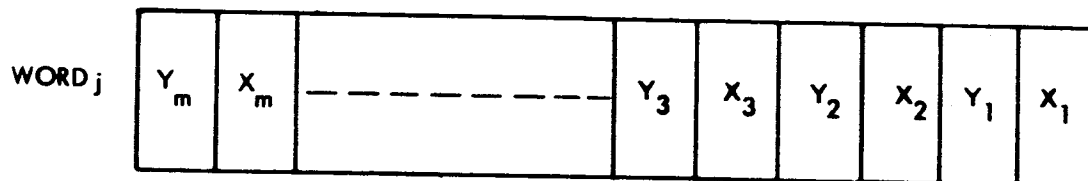
- a. Store the present contents of the Results Register in control bit C_1 .
- b. Perform a bit slice comparison.
 - 1) If one word satisfies the search, it is the maximum value
 - 2) If more than one word satisfy the search, return to step 1.
 - 3) If no words satisfy the search, read C_1 and then return to step 1.

It can be recognized that this procedure will be more time consuming than if two bits of storage were available external to the memory array since writing into the memory plane is undoubtedly slower than the transfer between two external registers.

Approach 2c - Cyclic Access -- This approach to an associative memory is based primarily on the capabilities of a class of devices. There are devices which are cyclic rather than random access. Such a device can be used to realize an associative memory fitting onto this category since it will have a bit slice writing capability. However there are a number of organizational descriptor differences. These are discussed below:

- a. There is no capability for doing word slice writing
- b. Word slice reading is not possible
- c. Access to bit slices is cyclic rather than random

The inability to do either reading or writing in the word slice direction does not effect the processing capabilities but does limit the practical areas of application. The inability to access bit slices randomly however does have an appreciable effect on the processing capabilities. Probably the greatest restriction is that all operations on a given set of words must be performed in the same bit sequence - starting at the most significant end, or starting at the least significant end. While the equality search and the inequality search can be carried out starting at either end, the maximum (minimum) search and the arithmetic operations have conflicting requirements. The maximum (minimum) search must proceed from the most significant end, while arithmetic operations must proceed from the least significant end. Arithmetic operations in general can be performed only if a complete serial adder per word is furnished. Even then the field add would not be possible unless they are interlaced rather than separated as shown below:



The lack of random access to bit slices rules out the use of control bits within the memory. Thus one of the main advantages of the bit slice writing capability is lost.

Other processing operations such as counting and shifting are equally difficult in this approach. Shifting, for instance, would require a complete cycle for each bit of shift.

Any of the cyclic access media could be used in this approach. Magnetic drums and magnetic discs offer the advantage of extremely low cost for very large memory sizes. They also employ non-destructive readout means and, even more important, are non-volatile thereby insuring the integrity of the memory if power should fail. The reliability problems of a rotating medium have led to the replacement of these types of memories by acoustic

delay lines, either of the nickel wire or glass types for many applications. The replacement process has been accelerated as the costs of these later types of memories (on a per bit basis) decrease. The glass delay lines constitute the only type of memory in which a search operation can proceed at a bit rate as high as 25 to 50 million bits per second. All of these delay lines are volatile devices and all require continuous circulation of the information. These are serious limitations for many memory applications. Research has been done on a type of magnetostrictive delay line called the strain wave memory which is not volatile and employs static storage. In this case the acoustic wave does not carry information and instead is used to reduce the coercive force of a magnetic film, which is plated on the wire, such that an applied field can then change the state of the film. Even NDRO operation can be achieved with this type of memory. Another non-volatile element in this category is the domain wall motion magnetic shift register which is attractive for medium speed memories with long bit length.

Semiconductor shift registers, particularly of the integrated circuit form, can also be used in this approach. The fastest of these would be capable of very high search rates of at least from 10 to 20 million bits per second, but power consumption would be prohibitive for very large memories. Tunnel diode shift registers might also be used in a similar manner, and in this case the speed would undoubtedly be limited only by the external bit slice logic. Shift speeds of 100 megabits per second or greater should be possible.

Approach 3 - Associative Memory with All-Parallel Equality Search

Description

This is the first organizational approach to utilize local logic to perform the search operations. The block diagram for this approach is shown in Figure 3-6. The most significant difference between this block diagram and those of the bit slice approaches discussed previously is that in this

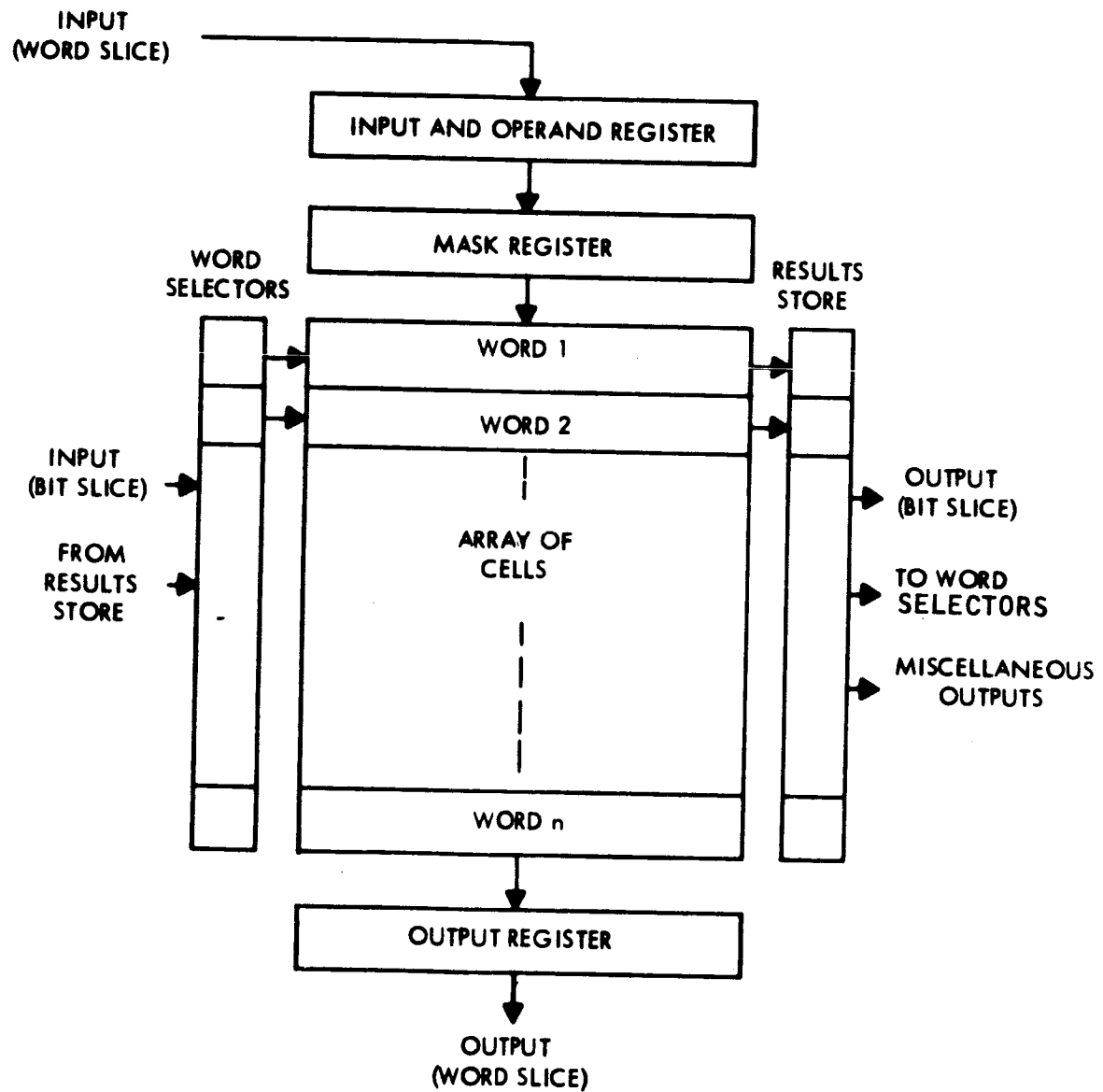


Figure 3-6. Approach 3 - Associative Memory With All-Parallel Equality Search

case the operand is furnished to the memory rather than the contents of the memory being read out to the External Logic. In fact the External Logic block has been deleted from the diagram.

The cell in this memory has in addition to the capability of storing a single bit of information, the capability of performing a simple logic function. The logic included in the cell, is the EXCLUSIVE-OR function; that is, the cell has the capability of performing the EXCLUSIVE-OR function between the stored bit and a bit furnished from the operand register. The output from each cell is a match-mismatch signal.

In order that local logic provide any advantages over external logic it is necessary that the signals be of such a form that an all-parallel equality search can be performed. If the cell is mechanized with magnetic devices the desired form is that the match signal be a null signal. Then the sense lead performs an OR function of mismatch signals as all bits are interrogated simultaneously. While for other types of mechanization the desired form of the signal may be different, a null match signal will be assumed in the discussion which follows.

Both bit slice and word slice writing capabilities are assumed for this approach. Bit slice and word slice reading are also included.

Capabilities

Equality Search -- The main advantage of local logic of the type present in this approach is the increased speed in performing an equality search. Since the match condition in each cell generates a null signal and the mismatch condition generates a non-null signal, an all-parallel approach is feasible. A mismatch condition in any one cell will produce the desired word mismatch

signal. A multiplicity of mismatch conditions in a given word will simply produce a larger mismatch signal. Note that if the match signal was not a null signal but simply opposite in polarity compared to the mismatch signal, an all parallel search would not be possible since match and mismatch signals would cancel each other out.

Inequality Search -- The all-parallel equality search capability allows the use of a faster algorithm⁴ for the inequality search than was possible in previous approaches. The number of cycles required for this algorithm is equal to the number of 1's or the number of 0's in the search word depending on whether a greater-than or less-than search is being performed. The procedure for a greater-than (less-than) search is as follows:

- a. Convert the least significant 0 (1) of the search word to a 1 (0).
- b. Masking out all bits of lesser significance than the converted bit, perform an equality search.
- c. The words satisfying the search are members of the set $S_G(S_L)$.
- d. Repeat steps 1 through 3 until all the 0's (1's) in the search word have been converted.

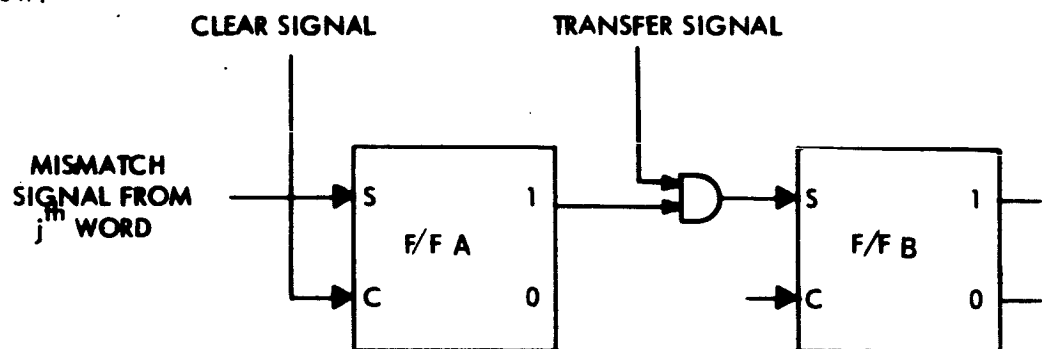
The union of all words satisfying these searches is the set of words greater than (less than) the search word.

An example of the alterations made to the search word in this procedure is given below

original	1 1 0 1 0 0 1
first search	1 1 0 1 0 1 X
second search	1 1 0 1 1 X X
third search	1 1 X X X X X

Note that a maximum of two stored words can satisfy the first search - 1 1 0 1 0 1 0 and 1 1 0 1 0 1 1. Both are greater than the original search word. The second search may locate as many as four additional words satisfying the search - 1 1 0 1 1 0 0, 1 1 0 1 1 0 1, 1 1 0 1 1 0, and 1 1 0 1 1 1 1. In general each search may locate 2^L stored words where L is equal to the number of masked out bits.

The organization of the Results Store facilities to mechanize this algorithm is shown below:



After each search, the B F/F's will be set if the A F/F contains a 1. The A F/F's are then all cleared in preparation for the next cycle. At the end of the search (all 0's have been converted to 1's) those B F/F's which are set correspond to words greater than (less than) the search word.

Maximum (minimum) Search -- This search is inherently a serial-by-bit operation and the all-parallel equality search does not provide any speed advantage over strictly bit slice searching. However there is an advantage in terms of the number of external storage locations required. This advantage is described in Approach 3b.

Proximity Search -- Although the proximity search is done by the same procedure described for Approach 2, the time required to perform it is less than that of Approach 2 due to an improvement in the counting operation.

Intersection and Union of Searches -- These searches are performed in the same way as described in Approach 2.

Field Addition ($X_1 + Y_1, X_2 + Y_2, \dots, X_n + Y_n$) -- The all-parallel equality search capability can be used to good advantage in performing the field add operation. The field add was previously defined as the operation of adding quantities X and Y stored in each word of the memory to obtain a set of sums $Z = X + Y$. The sum Z will be stored in place of the quantity Y since this would be the most frequently used type of field add. Another type, in which the sum Z is stored in a third field, can be performed also but it requires more time.

The algorithm to be described is due to Fuller ⁽⁴⁾ and uses a bit of the memory to store the carry bit, C . A truth table for this operation shows those combinations of X_i, Y_i, C_i which must be detected. Since the location of Y_i is used to store the sum bit Z_i and since the next carry C_{i+1} replaces the present carry C_i , the table can be presented as follows:

	<u>Present State</u>	<u>Next State</u>
	<u>$X_i \ Y_i \ C_i$</u>	<u>$X_i \ Z_i \ C_{i+1}$</u>
1.	0 0 0	0 0 0
2.	0 0 1	0 1 0
3.	0 1 0	0 1 0
4.	0 1 1	0 0 1
5.	1 0 0	1 1 0
6.	1 0 1	1 0 1
7.	1 1 0	1 0 1
8.	1 1 1	1 1 1

It can be noted that only input combinations 2, 4, 5, and 7 necessitate changes. It can also be noted that present state 4 generates a next state which is the same as present state 2, and present state 5 generates a next state which is the same as present state 7. Thus in order to prevent the occurrence of errors, input combination 2 must be processed before 4, and input combination 7 must be processed before 5.

If C is initially set to 0 and with the operation proceeding from least significant bit to most significant bit, the steps required for each bit of Z are as follows:

- a. Search for $X_i = 0$, $Y_i = 0$, and $C_i = 1$. Write $Z_i = 1$ and $C_{i+1} = 0$ in words satisfying the search.
- b. Search for $X_i = 0$, $Y_i = 1$, and $C_i = 1$. Write $Z_i = 0$ in words satisfying the search.
- c. Search for $X_i = 1$, $Y_i = 1$, and $C_i = 0$. Write $Z_i = 0$ and $C_i = 1$ in words satisfying the search.
- d. Search for $X_i = 1$, $Y_i = 0$, and $C_i = 0$. Write $Z_i = 1$ in words satisfying the search.

In summary this algorithm requires four searches and six write operations per bit of addition.

Addition ($X_1 + S$, $X_2 + S$, - - - - - $X_n + S$) -- The centralized add operation is carried out in a manner similar to that described for the field add. However in this case the quantity S is in the operand register rather than in the memory. Thus the searches are performed only over X and C, the particular combinations of the interest depending on whether S_i is a 1 or a 0. The operation requires two searches and three write operation per bit if Z_i replaces X_i . If Z_i is stored in a second field of each word the number of writes is the same as above but the number of searches is 2 or 3 depending on whether Y_i is 0 or 1, respectively.

Counting -- Counting in this associative memory is performed by using the same algorithm described for Approach 2. The all-parallel equality search allows the count to be made with two searches and 3 write operations per bit of count.

Other Processing Operation -- Shifting, complementing, and logic operations are performed in this associative memory in the same manner as described for Approach 2.

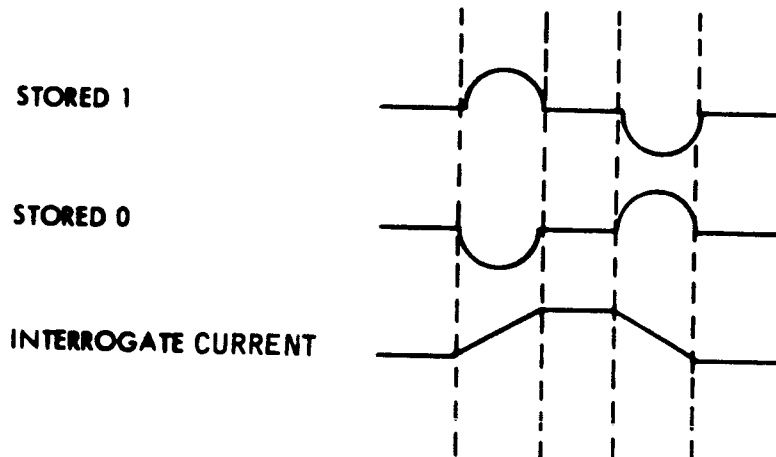
Mechanization Considerations

Requirements -- The cell used in this associative memory requires not only a storage function but also a logic function. Thus the device to be used in mechanizing the cell must provide both these functions. The fact that the stored bit is used internally (to the cell) in a logic operation means that the storage device must have a non-destructive readout (NDRO) capability and must be compatible with the logic performing elements.

Applicable Devices

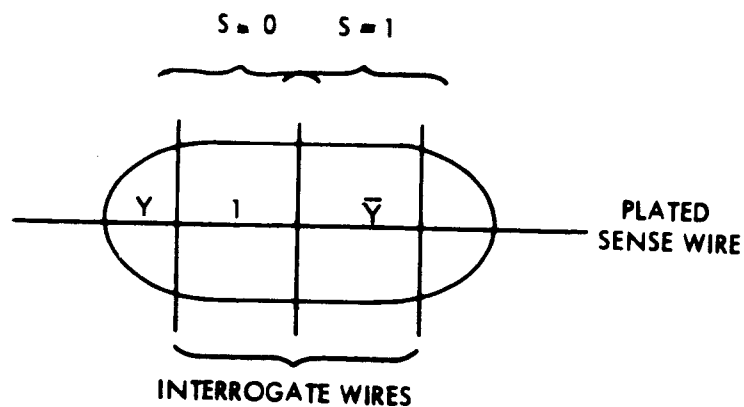
Plated Wire -- One device suitable for this approach is the plated wire element, which is described in detail in Appendix A. A cell made up of three plated wire elements can provide the capability of storing a single bit and of performing the EXCLUSIVE-OR logic function between the stored bit and a bit in the Operand Register.

When interrogated the plated wire element produces outputs of the following form:



Assuming that only the first polarity generated is recognized, the output produced when interrogating a stored 1 will be called a positive signal and that produced by a stored 0 a negative signal.

An associative memory cell is made up of three basic elements in the following configuration:



The notation within the cell indicates the quantity stored at each of the three storage elements.

Interrogation of the cell will result in the simultaneous interrogation of two of the three storage elements depending on the state of S.

This cell uses signal cancellation techniques to obtain the desired output which includes two possible signals - a null signal to indicate the match condition, and a positive signal for the mismatch condition. These are illustrated below:

X = 0, Y = 0	no signal
X = 0, Y = 1	positive signal
X = 1, Y = 0	positive signal
X = 1, Y = 1	no signal

This cell has the disadvantage that each bit must be stored in two locations. Thus bit slice writing requires two write cycles instead of one. A method of overcoming this disadvantage is described in Approach 4.

Integrated Circuits -- Integrated circuit flip-flops and logic elements can be used to provide the storage and logic requirements of this approach. For small memories this type of mechanization is feasible and would provide higher operating speeds than those attainable with magnetic elements. For large memories, line capacity limits the speed and the power consumption becomes impractically large. Field effect transistor circuits appear to be optimum for medium size (10,000 words), moderate speed memories while bipolar transistor circuits are better for small size (1000 words) high speed applications. It will undoubtedly be possible to place several bits (up to 10 or 20) on each chip thereby relieving some of the packaging and interconnection problems.

The cell, in this case, would consist of a transistor flip-flop with read-in logic and gated NDRO read-out plus an EXCLUSIVE-OR circuit that grounds the output line whenever the inputs are not equal. All of the EXCLUSIVE-OR circuits for a given word can then be connected

together and the output will be at ground potential if any of the bit pairs mismatch and will be at the higher potential only if all bit pairs match.

The integrated circuit cell, of course, is volatile, but it may be possible to alleviate this disadvantage. A ferroelectric storage element might be deposited on the integrated circuit chip and a circuit could be devised to automatically transfer the flip-flop contents into the ferroelectric capacitor whenever the power went off and back into the flip-flop as soon as power was restored. Ferroelectric capacitors are better than magnetic cores for this application since they can be driven and sensed by very simple circuits. It would not be desirable to leave the ferroelectric element in the circuit because of its low switching speed.

The cryotron is also suitable for very large memories of this type. Some of the advantages and disadvantages as well as the problems involved are discussed under Approach 4.

Variations of Approach 3

Approach 3a -- Approach 3a is simply Approach 3 without the bit slice writing capability. The main effect of this change is that it eliminates the capability of performing processing operations. These changes are noted in Table 3-1.

Approach 3b -- This variation of Approach 3 is obtained by reducing the external storage to a single bit per word. Such a change in this approach has less effect on the capabilities than in any of the previous approaches. The equality search and the processing operations are not effected at all and alternate schemes are available for the inequality and maximum (minimum) searches.

The inequality can be done by the same algorithm described for Approach 3 except that the partial results obtained after each equality search must be stored in a control bit of the memory. Another alternative is to use the same algorithm as described for Approach 1a. While the writing of results to the memory plane is not required in this algorithm, the number of searches is increased since a single bit search is required for each bit in the search field. The choice of algorithms depends on the write speed of the element.

The maximum (minimum) search is performed in this approach by using the all-parallel equality search to full advantage. The algorithm proceeds in a bit slice manner from the most significant bit to the least significant bit, but each successive search includes the next bit in addition to all the previous ones. Thus the first search will be over only the most significant bit, the second will be over the two most significant bits, the third over the three most significant, etc. The first time a bit position is included in a search, a 1(0) is contained in that position of the search register. However, if no words satisfy the search, the bit is changed to a 0(1) for all subsequent searches. This algorithm has the advantage that the words still in competition for maximum (minimum) value do not have to be remembered from one step to the next. Thus only a single bit of external storage is required.

Approach 4 - Local Logic - Ternary Output

Description

The logic function performed in the cell in this approach is slightly more complex than that of Approach 3. Sufficient logic must be provided in this case to separate the four possible combinations of two bits into three sets - thus a ternary output signal is required. This ternary output signal from the cell is desired for both the inequality search and for arithmetic operations.

The form of the ternary output signal will depend to a great extent on the mechanization of the cell. Although a number of devices can be used to mechanize such a cell, the signal assignment used here is aimed at the use of a magnetic device. The signal assignments are indicated in the table below:

Input Combinations	Output From Cell		
	Equality Search	Inequality Search	Add Operation
0 0	no signal	no signal	negative signal
0 1	positive signal	positive signal	no signal
1 0	positive signal	negative signal	no signal
1 1	no signal	no signal	positive signal

An additional requirement on this cell is that of performing a logic function on two bits stored within the memory. This is in contrast to the normal situation where one bit is stored in the cell and the other is furnished from the operand register. This requirement, which is useful for the field add operation, can be fulfilled either by providing for the storage of two bits within each cell or by allowing pairs of cells, each with a single bit of storage, to interact with each other.

To provide this variety of requirements a variable function cell is proposed, wherein the output of the cell varies depending on the operation being performed.

The block diagram of this approach is shown in Figure 3-7. It can be noted that bit slice and word slice modes are included for both the reading and writing operations.

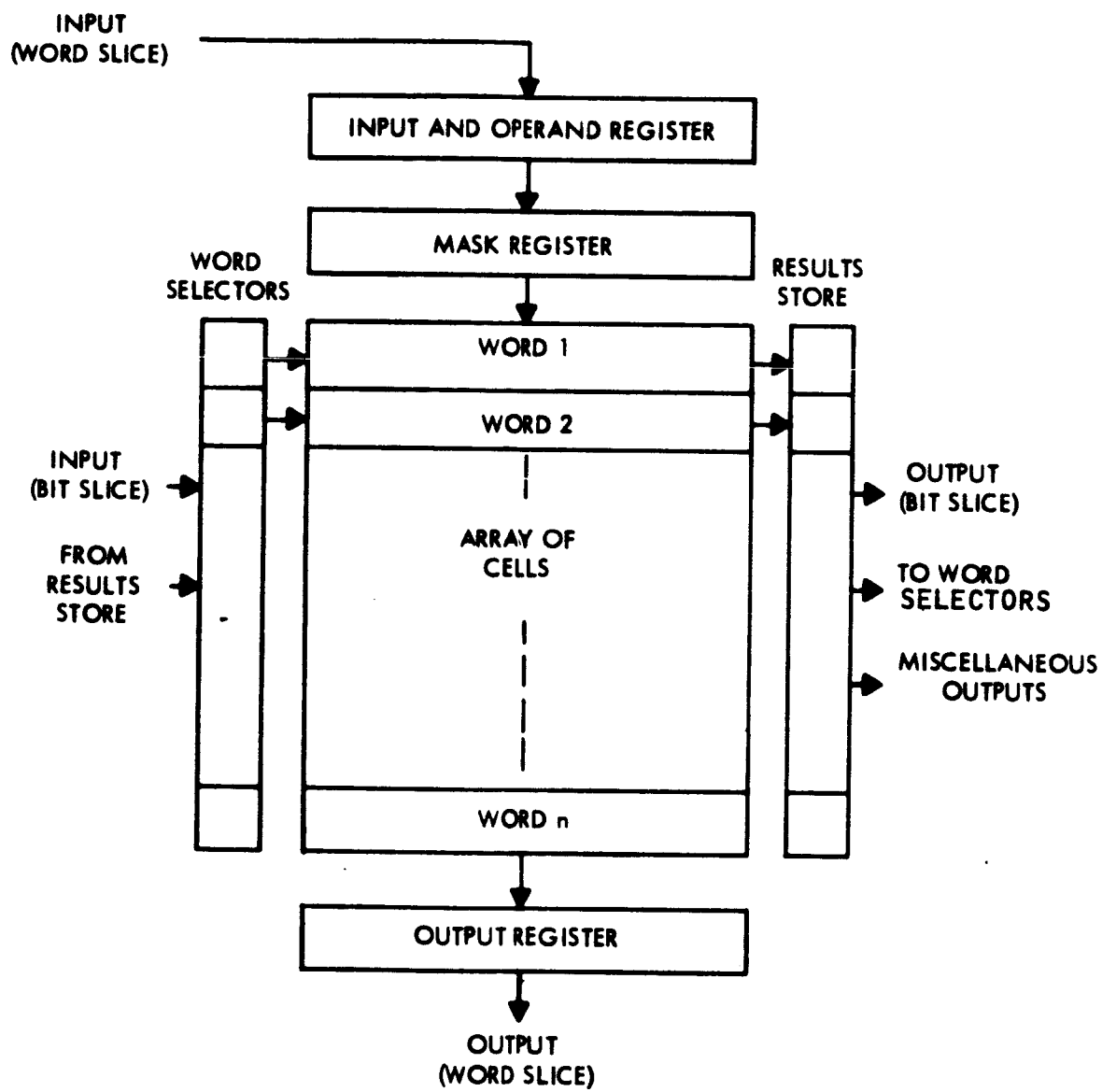


Figure 3-7. Approach 4 - Associative Memory With Local Logic - Ternary Output

Capabilities

Equality Search -- The cell specifications indicated that the outputs of the cell for the equality search operation will provide the null match signal and a positive mismatch signal. Thus an all-parallel equality search as described in Approach 3 is possible.

Inequality Search -- The output signal defined for the inequality search is a ternary signal which not only indicates the match - mismatch condition but also identifies the type of mismatch that has occurred. This allows a faster inequality search than was possible using external logic.

The algorithm proposed for use here is the same as previously described for Approach 1. The procedure is started by comparing the search word to stored words in the most significant bit position and proceeding in a bit slice mode. For a given word, when the first mismatch signal occurs, that word can be classified as less than or greater than the search word. If the assignment of signals is as defined above, a positive signal will indicate less than, and a negative signal will indicate the greater than condition. When the processing has proceeded through all bit slices, all words will have been separated into three sets - those less than, those greater than, and those equal to the search word.

With all the logic, needed to make the above decision, in the cell, no word to word synchronization is required as was the case with the external logic approaches. Because of this it is anticipated that the rate of processing of bit slices could be considerably faster in this approach than in previous approaches using external logic.

An additional speedup can be obtained by making a slight alteration in the algorithm. The presence of a null match signal allows the processing to be carried out by sequences of 1's and 0's rather than by a strictly bit slice

mode. For example if the search word is 0 0 1 1 1 0 1 1, a total of four search cycles are required rather than eight as in bit slice processing. The first two 0's can be searched simultaneously, then the three 1's can be searched simultaneously, then the next 0, and finally the last two 1's making a total of four searches.

Other Search Operation -- The maximum (minimum) search, the proximity search, and the intersection and union of searches are performed in the same way as described for Approach 2.

Addition ($X_1 + S, X_2 + S, \dots, X_n + S$) -- The requirements placed on the cell in order to perform the add operation without external logic were included in Table I. A ternary output signal is required which not only detects the match-mismatch condition but also differentiates between the two match combinations. The reason for this requirement can be seen by examining the Boolean expressions for the sum and carry bits written in a form to make maximum use of an EXCLUSIVE-OR logic capability.

$$Z_i = C_i (\bar{X}_i \bar{S}_i + X_i S_i) + \bar{C}_i (X_i \bar{S}_i + \bar{X}_i S_i)$$

$$= C_i \bar{A} + \bar{C}_i A$$

$$C_{i+1} = X_i S_i + (\bar{X}_i S_i + X_i \bar{S}_i) C_i$$

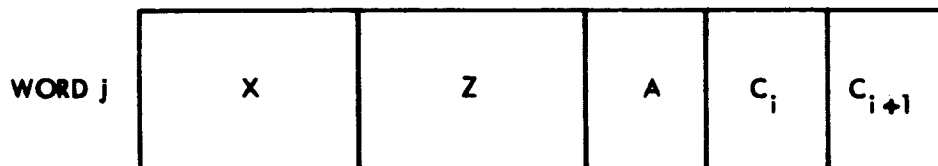
$$= X_i S_i + A C_i$$

$$\text{where } A = X_i \bar{S}_i + \bar{X}_i S_i$$

Note that in addition to the requirement for an EXCLUSIVE-OR function, there is a need for an AND function in the equation for C_{i+1} . The assignment of output signals from the cell (p. 3-55) provides the capability needed

by differentiating between the $X_i S_i$ and $\overline{X_i} \overline{S_i}$ combinations rather than between $X_i \overline{S_i}$ and $\overline{X_i} S_i$ as was needed for the inequality search.

Since no external logic is used, when the quantity A has been generated it must be stored in a control bit. Two control bits will also be used for the storage of carry bits — one for the present carry C_i and one for the next carry C_{i+1} . The anticipated allocation of bits within each word is shown below:



This makes obvious the need for performing the logic function on two bits stored within the word rather than between one stored bit and one bit furnished from the operand register as in all previously discussed operations.

The steps of the algorithm are outlined below:

Logic Operations

1. Perform logic on X_i and S_i

- | | | |
|----|---|----------------------------|
| a) | If output is positive, then $X_i = 1, S_i = 1$ | Write $A = 0, C_{i+1} = 1$ |
| b) | If output is negative, then $X_i = 0, S_i = 0$ | Write $A = 0, C_{i+1} = 0$ |
| c) | If output is "no signal" then $X_i = 1, S_i = 0$
or $X_i = 0, S_i = 1$ | Write $A = 1, C_{i+1} = 0$ |

2. Perform logic on C_i and A

- | | | |
|----|--|------------------------------|
| a) | If output is positive, then $C_i = 1, A = 1$ | Write $Z_i = 0, C_{i+1} = 1$ |
| b) | If output is negative, the $C_i = 0, A = 0$ | Write $Z_i = 0$ |
| c) | If output is "no signal", then $C_i = 1, A = 0$
or $C_i = 0, A = 1$ | Write $Z_i = 1$ |

In summary it can be seen that two searches (logic operations) and either three or four writes are required for a single bit add.

Field Addition ($X_1 + Y_1, X_2 + Y_2, \dots, X_n + Y_n$) -- The field add operation, in which both the X and the Y quantities are stored within the memory, is carried out in exactly the same way as the previous operation. It also makes no difference whether the sum Z is stored in a third field of each word or in place of either X or Y.

Count -- The count is performed in this memory by making use of the capability of performing a logic operation on two bits stored within each word of the memory. A single control bit is used to store the carry C_i and the field X is used to store the counter.

Assuming that the Results Store contains 1's in those positions corresponding to words to be incremented, the algorithm is as follows:

- a. Write the contents of the Results Store to control bit 1.
- b. Perform logic on X_i and C_i
 - 1) If output is positive, then $X_i = 1, C_i = 1$. Write $X_i = 0$
 - 2) If output is "no signal", then $X_i = 0, C_i = 1$ Write $X_i = 1, C_{i+1} = 0$
or $X_i = 1, C_i = 0$.
- c. Repeat step b for successively more significant bits of X.

Thus a single read (logic operation) and three writes are required for each bit of the counter.

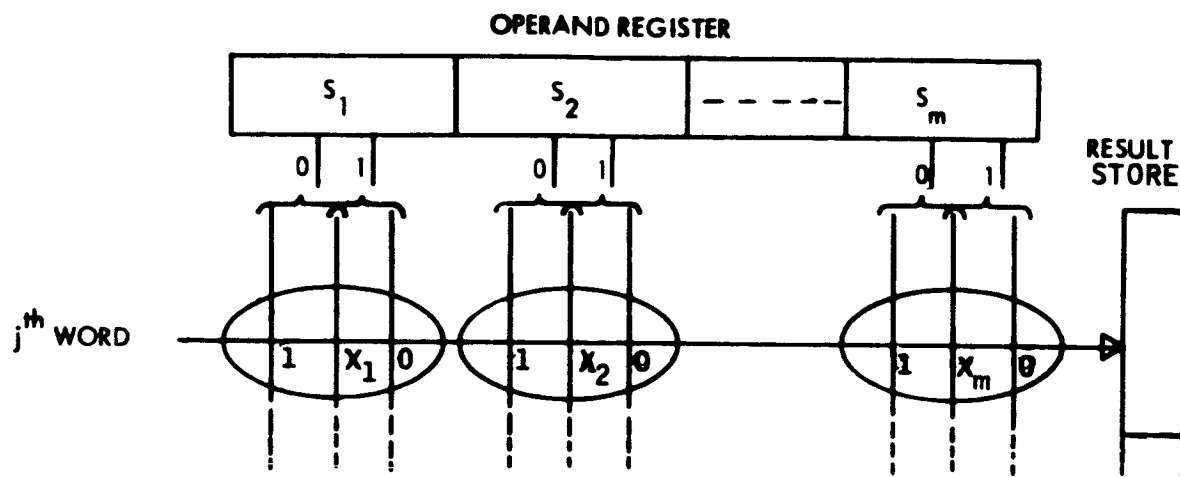
Other processing operations -- Shifting, complementing, the logic sum, and the logic product are performed in essentially the same manner as described for Approach 2.

Mechanization Considerations

Requirements -- The device requirements are probably more severe for this organizational approach than any of the previous ones. Since the logic function is distributed throughout the memory, the device must have both logic and storage capabilities. This plus the need for a null match signal are no different than the requirements of Approach 3. The additional features needed in this cell are a ternary output, where the assignment of minterms to output signal varies depending on the operation being performed, and the capability of performing the logic function on two stored bits as well as on one stored bit and one furnished from the operand register.

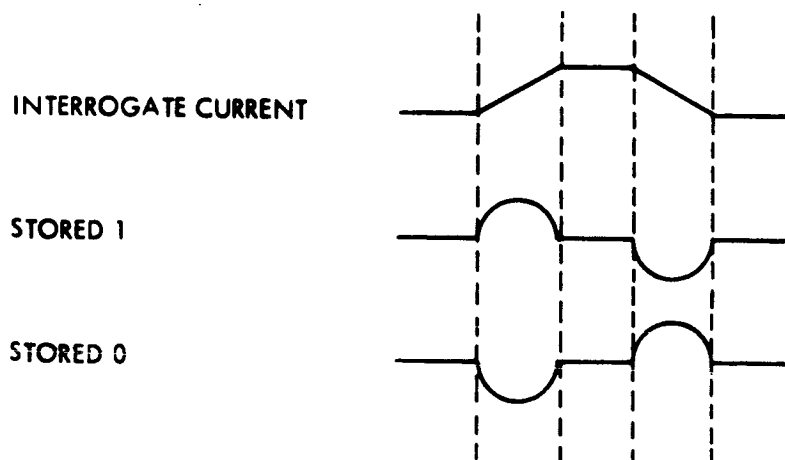
The bit slice writing capability is again quite important as it is in any approach aimed at processing rather than just searching.

Applicable Devices -- Surprisingly not all magnetic devices can be immediately ruled out of this organizational approach. The plated wire memory element, operating in a mode of operation similar to that described for Approach 3, appears to be a feasible device. The mode of operation utilized here is signal cancellation of the same general type as previously described. A set of three of the plated wire storage elements are grouped together to form what is called a SCANCELL (Signal Cancellation Cell). In addition to performing a storage function the cell can perform the desired logic functions required for the equality search, for the inequality search, and for the arithmetic operations. The way this is done is shown below where X and Y are stored quantities and S is the contents of the operand register:

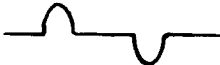
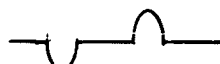


The notation implies that two interrogate wires of each cell are energized at a time — the left two if the bit of the Operand Register contains a 0, and the right two if it contains a 1. The quantities X , 1, and 0 are stored at the three intersections of the plated wire and interrogate wire occurring in each cell as shown above.

Consider first the logic function performed by the cell for an inequality search. Assuming that the output from each storage element is

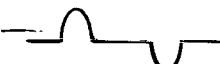



and that adjacent elements are sufficiently uniform so that a 1 and 0 interrogated simultaneously will cancel each other out, the following output signals are obtained.

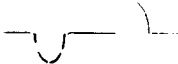
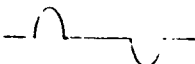
<u>Input Combination</u>		<u>Output</u>
<u>S</u>	<u>X</u>	
0	0	No signal
0	1	
1	0	
1	1	No signal

If we assume that only the first position of the bi-polar output is recognized, this is seen to satisfy the requirements defined for the inequality search.

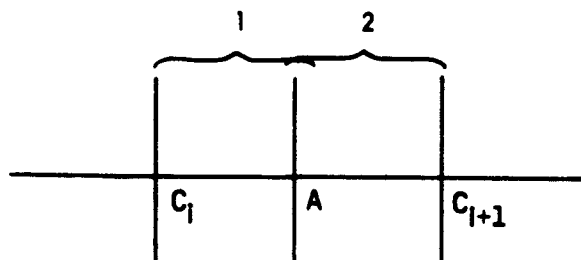
This same cell can produce the outputs required for an all-parallel equality search by simply turning on the interrogate current in those bit positions of the search word containing 1's prior to the search operation. The actual search is performed by simultaneously turning off interrogate currents corresponding to 1's in the search word and turning on interrogate currents corresponding to 0's in the search word. This will produce the following:

<u>Input Combinations</u>		<u>Output</u>
<u>S</u>	<u>X</u>	
0	0	No signal
0	1	
1	0	
1	1	No signal

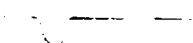
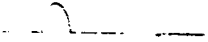
Another slight change will provide the logic function required for arithmetic operation. By simply inverting the contents of the operand register prior to interrogation and by performing the interrogation as for the inequality search, the following logic function is performed:

<u>Input Combination</u>		<u>Output</u>
<u>S</u>	<u>X</u>	
0	0	
0	1	No signal
1	0	No signal
1	1	

This can be seen to be one of the logic function requirements for arithmetic. The other one was the generation of the same set of outputs with two stored quantities, A and C, serving as the inputs. The use of the three storage locations within the cell to perform this function is as shown below:

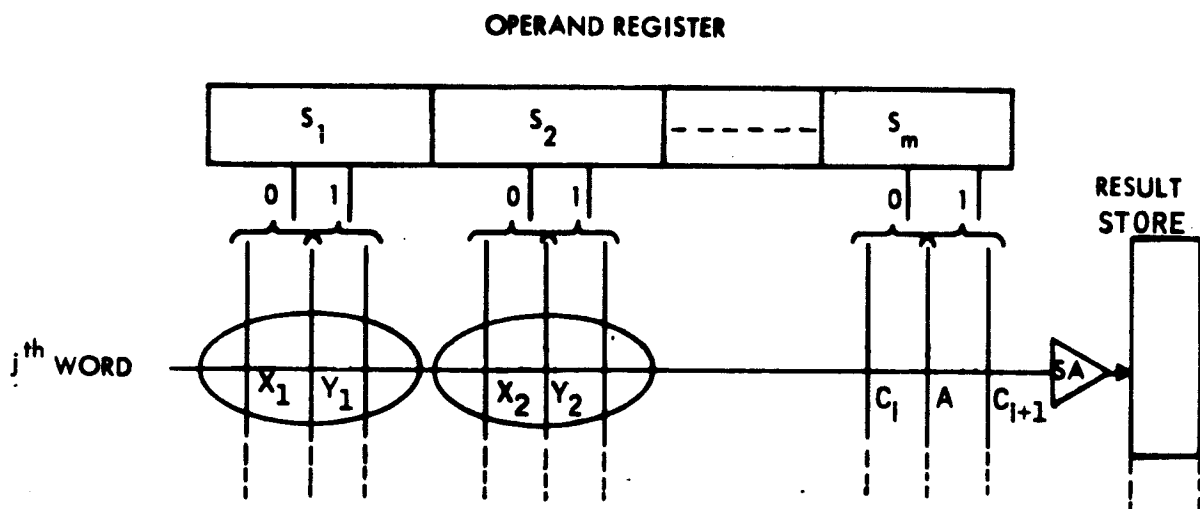


The logic is performed by simply interrogating pair 1 or pair 2 depending on which is the appropriate C. The signals generated are

<u>C</u>	<u>A</u>	<u>Output</u>
0	0	
0	1	No signal
1	0	No signal
1	1	

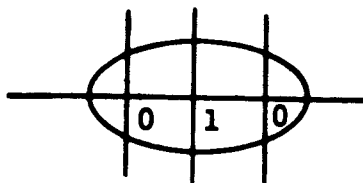
which is the desired output.

The field add can also be performed, with allocation of cells and storage elements within the cell as follows:



The Operand Register is initially set to 0 with S_m alternating between 0 and 1 for each bit of addition performed.

The SCANCELL provides another unique and useful capability — that of storing a don't care. With a don't care stored in the cell, regardless of which pair of storage elements is interrogated, the output of the cell is "no signal". The storage of a don't care is shown below:



This capability effectively allows the masking out of selected bits on a per word basis, rather than simultaneously for all words as is done by using the mask register. Such a capability is useful in some types of pattern recognition problems.

Integrated circuits can be used in this approach much as in Approach 3. The cell becomes slightly more complicated but otherwise the technique is very similar. Similar advantages occur for small size memories and similar power penalties must be paid. Cryotrons can also be used as in Approach 3 and with similar advantages and problems.

Approach 5 - Intercommunicating Cells

Description

The next step in increased capabilities in an associative memory is to provide the capability of passing results directly from one cell to the next. To fully utilize this intercommunicating capability, the cell contains the logic necessary to perform an add operation. This logic can also provide the type of output needed for an all-parallel equality search and

the ternary output signal needed for the inequality search. The full adder per cell and intercommunicating cells seem to go hand in hand as will be seen in the description of arithmetic capabilities.

A block diagram of this approach is shown in Figure 3-8. The array of such cells can be written-into and read-from in either the bit slice or word slice modes. Also since each cell has a number of different functions to perform, command information must be distributed to each cell. In all previous approaches the command information was used only on the periphery of the array since the cell performed the same way for all operations. The list of commands might include store, read, compare, add, shift (right or left), and complement. The commands are assumed to be provided simultaneously to all cells of a column. Of course any row of cells can be masked out for a given command.

Each internal cell in the array can communicate directly with its two nearest neighbors in the row. However, additional intercommunication can be obtained by using the row (word) output line as an input line to the cell for certain commands. For instance consider the requirement to shift the contents of the right-most column to the left-most column in Figure 3-8. This can be done in one operation by supplying the read command to all cells in the right column and a "special store" command to cells in the left column. This "special store" command will cause the cells to store the contents of the row output line. This capability is also useful for other operations.

Capabilities

Equality, Inequality, and Maximum (minimum) Searches -- Since the ultimate in speed of performance of these searches was achieved in previously described approaches, the cellular intercommunications capability provides no improvements.

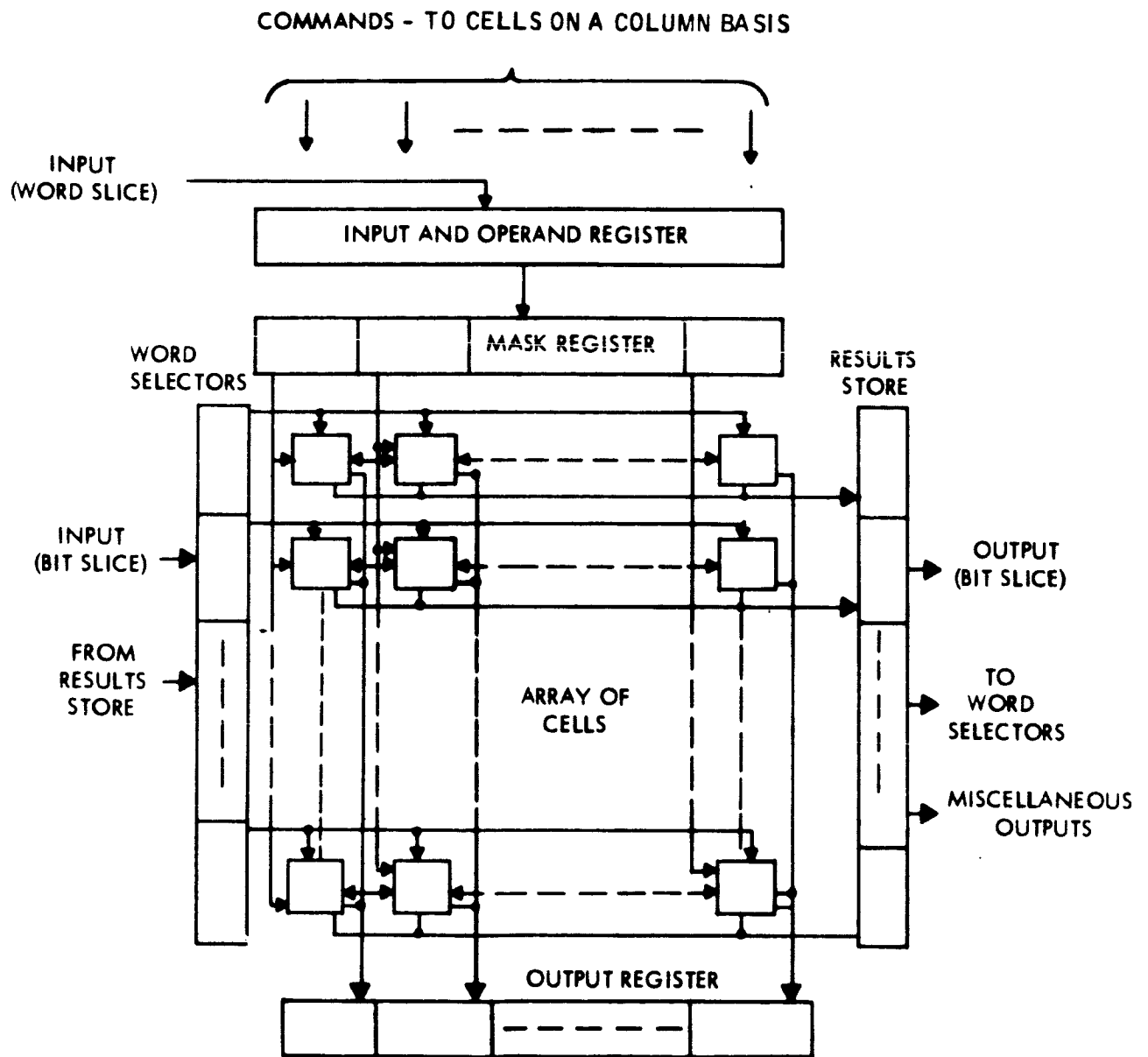


Figure 3-8. Approach 5 - Associative Memory With Intercommunicating Cells

Proximity Search -- This search must again be carried out by counting the number of mismatches and performing a minimum search over the final counts. Thus the improvement in performing this operation is directly proportional to the improvement in counting.

Intersection of Searches -- The intersection of searches is again performed in the same way as in Approach 2.

Union of Searches -- The capability of using any column of cells within the array to receive the results of a search operation on other columns provides an advantage for a union of searches operation. This effectively allows the use of any column as a Results Store and thus eliminates the need for transfer from the results store to the array as was required in all other approaches.

Addition ($X_1 + S, X_2 + S, X_3 + S, \dots, X_n + S$) -- The use of the intercommunications capability for the add operation is to propagate the carry signal from the least significant end. The contents S of the operand register are simply applied to the array and those words which are active will proceed to carry out the add by starting at the least significant end. Each cell, having received the operand bit and the carry-in from the neighbor to the right, will generate a sum bit Z which will replace the stored bit X , and will generate a carry bit to be transmitted to the neighbor on the left. Thus the operation is performed as rapidly as the carry can propagate.

It can be noted that cellular intercommunications from right to left only is sufficient for this operation.

Field Addition ($X_1 + Y_1, X_2 + Y_2, \dots, X_n + Y_n$) -- This operation is carried out by making use of the row (word) output line as an input line to transfer the i^{th} bit of Y to the cell containing X_i . The add operation, $X_i + Y_i \rightarrow Z_i$, is performed with the sum bit Z_i replacing X_i and the carry C_{i+1} being transferred to the cell on the left.

Counting -- Counting is performed in the same way as the centralized add where the operand will contain a single one in the position corresponding to the least significant bit of the counter, and 0's elsewhere.

Shifting -- Assuming that the storage element is the equivalent of a double rank flip-flop, the shift operation is performed by simply providing the proper shift command to the columns to be shifted. A shift left command, for example, will cause the cell to output its contents to the cell on its left and to store the input received from the cell on its right.

Complement -- It is assumed that a cell would have the capability of complementing its contents on command, thus avoiding the reading and re-writing required in previous approaches.

Mechanization Considerations

Requirements -- The cells in this approach have the capability of storing a single bit of data and also include a full adder. The most severe requirements, however, is that the cells be able to communicate with each other.

Device Considerations -- The addition of intercommunication between cells to the local logic associative memory constitutes a major change. The bit-iterative operations that depend on a carry propagation no longer are speed limited by the bit access time - the speed of the local logic elements is the prime factor in this case.

Integrated circuit elements and cryotrons can again be employed in a similar manner to that of Approaches 3 and 4. Since the carry propagation time of cryotron circuitry depend on a time constant rather than on the switching speed of cascaded elements, the operating speed of a cryotron associative memory should be about the same as that of an integrated circuit memory - possibly even greater.

In this approach circuit elements such as the tunnel diode would be capable of realizing their inherent speed capabilities. This was not possible in the previous approaches because of the speed limitations of the accessing circuitry. Many types of logic circuits employing tunnel diodes have been proposed and at least one of these should be suitable for memory applications. Attempts to integrate tunnel diodes on monolithic chips have recently met with some success. This would tend to reduce the high cost of these devices and possibly increase the uniformity of characteristics. In either the discrete element or integrated circuit case, the size of a tunnel diode memory and that of integrated circuit bi-polar transistor would be restricted to small size units; larger memories would probably be constructed with integrated circuit field effect transistors or cryotrons.

The major hope for very large (greater than 10,000 words) associative memories with local logic and intercommunicating cells lies in the cryotron. Since the cryotron exhibits true zero resistance in the superconducting state and a very low resistance in the normal state, the total power dissipation is extremely low. The small element size and direct compatibility of memory and logic elements permits extremely complicated logic structures to be included in each cell. Either of two types of memory elements can be employed - the cryotron flip-flop or the persistent current memory element. The latter type is non-volatile in a certain sense; the memory contents is not destroyed when the logic power fails as long as the temperature of the memory remains within a certain range. The speed

of a cryotron memory should be in the same range as that of magnetic memories, i. e. , read speeds of 0.1 to 5.0 microseconds depending on size. Production difficulties have so far prevented the economical construction of cryogenic memory planes, but the recent application of integrated circuit construction techniques appears to have solved some of these problems. Other problem areas involve the interface with the outside environment and the economics of maintaining cryogenic temperatures.

SUMMARY OF ORGANIZATIONAL APPROACHES

The associative memory organizational approaches described in this section are summarized in Table 3-1. Each approach is described in terms of a set of computation descriptors, in terms of its searching and processing capabilities, and in terms of the devices suitable for its mechanization.

The entries used in the capabilities portion of the table are to give an indication of the relative capability of each organizational approach for each of the searching and processing operations. The larger the number the faster an operation can be performed. It can be noted that the largest number is a five, which is the entry for all operations for Approach 5. Thus all other associative memories are rated relative to Approach 5. The device used to mechanize a given approach has no effect on this number; it is rather effected primarily by the algorithm that must be used to carry out the operation.

The three entries used in the devices part of the table are an indication of the size associative memory that could be realized by a given device. A small memory of 1,000 words or less is designated by an S, a medium memory from 1,000 to 10,000 words by an M, and a large memory of more than 10,000 words by an L. The absence of an entry indicates that

Table 3-1. Summary

ORGANIZATIONAL APPROACHES		ORGANIZATIONAL DESCRIPTORS						Search Operations				
		Cell Characteristics	Writing Mode	Reading Mode	Access for Processing	External Logic per Word	External Storage per Word					
		Single Bit of Storage Local Logic-Binary Output Local Logic-Ternary Output Variable Function Command Required Full Adder	Word Slice Bit Slice	Word Slice Bit Slice	Cyclic Random	Exclusive OR Serial Adder	Single Bit Multiple Bits					
Minimum Associative Memory	1 1a 1b	X X X	X X	X X X X X X	X X X	X X X	X X X	2 2 1	2 2 1	5 5 5	1 1 3	5 5 3
Associative Memory with bit slice writing	2 2a 2b 2c	X X X X	X X X X X X X	X X X X X X X	X X X X	X X X X	X X X X	2 2 1 1	2 2 3 1	5 5 2 1	3 3 2 1	5 5 5 5
Associative Memory with All-Parallel Equality Search	3 3a 3b	X X X X X X	X X X X X X	X X X X X X	X X X	 	X X X	5 5 5	4 4 3	5 5 5	3 1 2	5 5 5
Associative Memory with Ternary Output from Cell	4	X X X X	X X	X X	X		X	5	5	5	3	5
Associative Memory with Intercommunicating Cells	5	X X X X X X	X X	X X	X		X	5	5	5	5	5

3-73-1

7 of Organizational Approaches

CAPABILITIES						DEVICES													
Processing Operations			Complex Operations	Active Elements		Hysteresis Elements										Read Only Elements		Cyclic Access Media	
				Integrated	Discrete														
Add (X ₁ + S, X ₂ + S, --X _n + S) Add (X ₁ + Y ₁ , X ₂ + Y ₂ , --X _n + Y _n) Count Shift Complement Logic Sum and Product			Proximity Search Floating Point Add Summation (X ₁ + X ₂ + --- + X _n)	Monolithic Bipolar MOST Thin-Film Field Effect Tunnel Diodes	Cryotrons Tunnel Diodes											Slug Inductive Coupling E Core Diode	Integrated Circuit Shift Register FET Shift Register Delay Lane Strain Wave Thin Film Shift Register		
				S S	S S	ML ML ML ML ML ML ML ML ML ML ML ML ML ML ML ML ML ML ML													

a device would probably not be considered for that organizational approach. For instance the monolithic bipolar integrated circuit is not considered for the minimum associative memory, not because it couldn't satisfy the requirements, but rather because it fits the requirements of an associative memory employing local logic much better. It can also be noted that not all the devices described in the survey of Appendix B are included. Devices were excluded from this table for one of two reasons—the device was given a confidence level of 6, 7, or 8 in the survey, or the device did not appear to compare favorably with other available devices for any of the organizational approaches.

BIBLIOGRAPHY FOR SECTION III

Magnetic Associative Memories

1. J. R. Kiseda, H. E. Petersen, W. C. Seelbach, and M. Teig, "A Magnetic Associative Memory," IBM J. Res. Dev., Vol. 5, pp. 106 - 121; April 1961.
2. R. R. Lussier and R. P. Schneider, "All-Magnetic Content Addressed Memory," Electronic Industries, p. 92; March, 1963.
3. C. A. Rowland and W. O. Berge, "A 300 Nanosecond Search Memory," Proc. Fall Joint Computer Conf., Vol 24, pp. 59 - 65; November 1963.
4. A. D. Robbi and R. Ricci, "Transfluxor Content-Addressable Memory," Proc. Internat'l Conf. on Nonlinear Magnetics, ; April, 1964.
5. J. McTeer, J. Capobianco, R. L. Koppel, "Associative Memory System Implementation and Characteristics (Biax), " Proceedings of Fall Joint Computer Conference, 1964.
6. Capt. Joseph Lucas, (USAF) "Development of a 24 Thousand Bit Associative Memory Utilizing Passive Detection," 1964.

Cryogenic Associative Memories

1. A. E. Slade and H. O. McMahon, "A Cryotron Catalog Memory System," Proc. Eastern Joint Computer Conf. , pp. 115-119; December, 1956.
2. R. R. Seeber, "Cryogenic Associative Memory, "presented at the National Conference of the Association for Computing Machinery, Milwaukee, Wisc.; August, 1960.

3. A. E. Slade and C. R. Smallman, "Thin-Film Cryotron Cryotron Catalog Memory," IEEE Trans. on Automatic Control, Vol. 13, pp. 48-50; August, 1960
4. H. T. Mann and J. L. Rogers, "A Cryogenic 'Between-Limits' Associative Memory," Proc. Nat'l Aerospace Electronics Conv., 359; May 1962.
5. V. L. Newhouse and R. E. Fruin, "A Cryogenic Data Addressed Memory," Proc. Spring Joint Computer Conf., Vol. 21 pp. 89-93; May, 1962.
6. P. M. Davies, "A Superconductive Associative Memory," Proc. Spring Joint Computer Conf., Vol. 21, pp. 79-88, May, 1962.
7. R. W. Ahrons and L. L. Burns, Jr., "Superconductive Memories," Computer Design, Vol. 1, pp. 12-19; January, 1964.
8. Richard Ahrons, "Superconductive Associative Memories," RCA Review Sept. 1963.
9. J. P. Pritchard, Jr. and L. D. Wald, "Design of a Fully Associative Cryogenic Data Processor", Proc. International Conf. on Nonlinear Magnetism, April, 1964.

Semiconductor Associative Memories

1. R. C. Corbell, (UCLA) "A Tunnel Diode Associative Memory," M. S. Thesis, University of California at Los Angeles; 1962.
2. M. H. Lewin, H. R. Beelitz, and J. A. Rajchman, "Fixed Associative Memory Using Evaporated Organic Diode Arrays," Proc. Fall Joint Computer Conf., Vol. 24, pp. 101-106; November 1963.

3. E. S. Lee, "Associative Techniques with Complementing Flip-Flops," Proc. Spring Joint Computer Conf., Vol. 23, p. 381; May, 1963.
4. E. S. Lee, "Solid State Associative Cells," Pacific Computer Conference, 1963.

Associative Memory Systems

1. E. H. Frei and J. Goldberg, "A Method for Resolving Multiple Response in a Parallel Search File," IRE Trans. on Electronic Computers, Vol. EC-10, pp. 718-722; December 1961.
2. C. Y. Lee and M. C. Paull, "A Content Addressable Distributed Logic Memory with Applications to Information Retrieval," Proc. IEEE, Vol. 51, pp. 924-932; June, 1963.
3. Collection of "Technical Notes on Goodyear Associative Memory," Goodyear Aircraft Corp., Akron, Ohio, Report No. GER 10857, October, 1962.
4. G. T. Tuttle, "How to Quiz a Whole Memory at Once," Electronics, Vol. 36, pp. 43-46; November 15, 1963.
5. H. Weinstein, "Proposals for Ordered Sequential Detection of Simultaneous Multiple Responses," IEEE Trans. Electronic Computers (correspondence), Vol. EC-12, pp. 564-567; October, 1963.
6. R. H. Fuller, (UCLA) "Content Addressable Memory Systems," University of California at Los Angeles, DDC AD 417644; June, 1963.
7. A. Kaplan, "A Search Memory Subsystem for a General-Purpose Computer," Proc. Fall Joint Computer Conf., Vol. 24, pp. 193-200; November, 1963.

8. "Summary of Investigation on Associative Memories," Contract 4068 (00) ONR, 15 January 1964.
9. Yoahan Chu, "A Destruction Readout Associative Memory," IEEE Transactions on Electronic Computers, August 1965.
10. L. R. Johnson and M. H. McAndrew "On Ordered Retrieval from an Associative Memory," IBM J. Res. Div., Vol. 8, pp. 189-193; April 1964.

Associative Memory Algorithms

1. R. R. Seeber, "Associative Self-Sorting Memory," Proc. Eastern Joint Computer Conf., Vol. 18, pp. 178-187; December, 1960.
2. R. R. Seeber and A. B. Lindquist, "Associative Memory with ordered Retrieval," IBM J. Res. and Dev., Vol. 6, pp. 126-136; January, 1962.
3. M. H. Lewin, "Retrieval of Ordered Lists from a Content Addressed Memory," RCA Review, Vol. 23, pp. 215-229; June, 1962.
4. A. D. Falkoff, "Algorithms for Parallel - Search Memories," J. ACM, Vol. 9, pp. 488-511; October, 1962.
5. G. Estrin and R. Fuller, (UCLA) "Algorithms for Content-Addressable Memories," Proc. Pacific Computer Conf.; March, 1963.
6. Rogers and Wolinsky, "Associative Memory Algorithms and Their Cryogenic Implementation," AD 429521, December, 1963.

Applications of Associative Memories

1. R. R. Seeber, "Symbol Manipulation with an Associative Memory," Preprints Nat'l Conf. Association for Computing Machinery, pp. 5B-4 (1) through 5B-4 (4); September, 1961.
2. L. Bloom, C Cohen, and S. Porter, "Considerations in the Design of a Computer with a High Logic-to-Memory Speed Ratio," Gigacycle Computing Systems, AIEE Spec. Pub. S-136, pp. 53-63; 1962.
3. E. C. Joseph and A. Kaplan, "Target Track Correlation with a Search Memory," Proc. 6th Nat'l. Conv. on Military Electronics, p. 255; June, 1962.
4. T. Singer and P. Schupp, "Associative Memory Computers from the Programming Point of View," MITRE Corp., Bedford, Mass., Rept. No. W-5492; August, 1963.
5. R. H. Fuller and G. Estrin, (UCLA) "Some Applications for Content-Addressable Memories," Proc. Fall Joint Computer Conf., Vol. 24, pp. 495-508; November, 1963.
6. R. R. Seeber and A. B. Lindquist "Associative Logic for Highly Parallel Systems," Proc. Fall Joint Computer Conf., Vol. 24, pp. 489-493; November, 1963.
7. M. E. Conway, USAF (ESD) "A Multiprocessor System Design," Proc. Fall Joint Computer Conf., Vol. 24, pp. 139-146; November, 1963.
8. "The Application of Associative Memories to Control Functions in a Multi-Processor Computer System," pp. 4-39 to 4-66, MILDATA Quarterly Progress Report No. 3 by Honeywell, Contract No. DA-36-039-AMC-03275 (E), June, 1964.

9. Yaohan Chu, "Application of Content-Addressed Memory for Dynamics Storage Allocation, " RCA Review, March, 1965.

Associative Processors

1. R. F. Rosin (University of Michigan), "An Organization of an Associative Cryogenic Computer, " Proc. Spring Joint Computer Conf. , Vol. 21, pp. 203-209; May, 1962.
2. P. M Davies "Design for an Associative Computer, " Proc. Pacific Computer Conf. , p. 109;, March, 1963.
3. J. Bernard F. Behnek, A. Lindquist and R. Seeber, " Structure of a Cryogenic Associative Processor, " Proceedings of IEEE, October, 1964.
4. J. P. Pritchard, Jr. and L. D. Wald, "Design of a Fully Associative Cryogenic Data Processor, " Proc. Internat'l Conf. on Non-Linear Magnetics, pp. 251-254; April, 1964.
5. Ewing, P. Davies, "An Associative Processor, " Proceedings of the Fall Joint Computer Conference, 1964.
6. R. M. Bird and R.H. Fuller, "An Associative Parallel Processor with Application to Picture Processing, " Proceeding of the Fall Joint Computer Conference, pp. 105, November, 1965.

REFERENCES FOR SECTION III

1. A. E. Slade and H. O. McMahon, "A Cryotron Catalog Memory System," Proc. Eastern Joint Computer Conference, pp. 115-119, December, 1956.
2. "Summary of Investigation on Associative Memories," Computer Command and Control Company, Contract No. 4068 (00) ONR, 15 January, 1964.
3. "Collection of Technical Notes on Goodyear Associative Memory," Goodyear Aircraft Corp., Akron, Ohio, Report No. GER 10857, October, 1962.
4. R. H. Fuller, "Content Addressable Memory Systems," University of California at Los Angeles, DDC AD 417644, June, 1963.

SECTION IV

INVESTIGATION OF SPECIAL APPLICATIONS AND APPROACHES

The computation requirements for unmanned space vehicles were summarized in Section II. A number of conclusions were drawn which point to problem areas where associative memories can be expected to be applicable. Taking these conclusions into consideration, a number of subtasks have been identified and are listed below:

1. Investigate the use of an associative memory for incremental computation such as is required in navigation and control functions.
2. Consider the use of associative techniques for compression of pictures. A two-level associative memory looks promising for this task.
3. Investigate associative techniques to provide an adaptive sampling capability for both analog and digital quantities.
4. Consider the organization of an associative memory aimed at the sum of products computation.
5. Investigate the use of associative techniques to handle (executive) control functions in a multi-processor system. In this case it is assumed that the on-board computer facility is a multi-processor system, which is reasonable in view of the reliability requirements.
6. Consider methods of making an associative memory ultra-reliable.

Only items 1 and 6 have been examined to any extent at this time. Discussion of these two subtasks follows:

AN ASSOCIATIVE MEMORY FOR INCREMENTAL COMPUTATION

It has been assumed that a strapped down attitude reference system employing a set of laser gyros would be used as the basis for the navigation and control system for the lander. Most of the computations that must be performed by this system are of the incremental type, many of which must be performed at a high rate. Fortunately, most of these computations can be done in parallel rather than sequentially and it is therefore practical to construct a series-parallel special purpose incremental computer to do this job.

It may however be desirable to use an associative memory to do these computations. First of all, although the associative processor would probably contain more elements than an incremental computer designed to do the same job, the nature of the associative processor - a large network of identical elements - may make a low cost batch fabrication process possible. In that case the associative memory may be less expensive than the incremental computer. Furthermore, if there is to be an associative memory in the lander for the purpose of processing data obtained during the performance of scientific experiments after the landing phase, it would be available for the navigation and control function during the landing phase.

Incremental computation is usually performed by suitably interconnecting a number of digital integrators. Each integrator computes according to the formula $\Delta Z = k Y \Delta X$ which when summed forms $\sum \Delta Z = Z = k \sum Y \Delta X$. This sum is an approximation to the integral $Z = k \int y dx$.

The digital integrator consists of three major parts - a Y register connected as a counter, an R register and adder-subtractor connected as an accumulator, and the necessary logic for generating the ΔZ and controlling the counter and accumulator from the ΔX and ΔY inputs. Figure 4-1 is a block diagram of a typical integrator configuration. Although binary (two valued) increments are

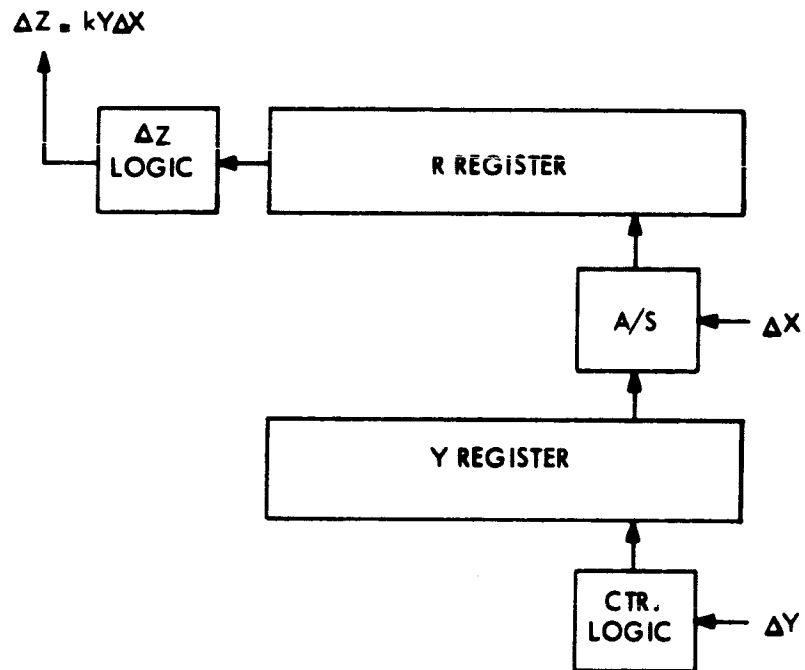


Figure 4-1. Block Diagram of an Integrator for an Incremental Computer

sometimes used, ternary (3 valued) increments are much more common and will be assumed in this application. The three possible incremental values are + 1, 0, and - 1. Means must be provided to add several ΔY increments into the contents of the Y register. This can be done either sequentially or in parallel. Addition of a single increment to the Y register requires counting up by 1, no change, or counting down by 1. Addition of more than one increment may require several counts. The R register is updated by either adding Y to R, leaving the contents of R unchanged, or subtracting Y from R depending on the value of ΔX . If this is done after the Y register is updated with the most recent ΔY , the new value of Y is used; if it is done before the Y register is updated, the old value of Y is being employed. The ΔZ increment consists merely of the overflow or output carry (or borrow) of the R register. It is sensed after each $R \pm Y$ operation and then encoded and stored for later use as either a ΔX or ΔY input to other integrators.

Scaling consists of determining the proper values for the constant k associated with each integrator and modifying the integrator such that the output $\Delta Z = kY \Delta X$. If k can be made equal to a power of two, the scaling process can be implemented by selecting the proper position in the X register in which to add the ΔY increment. In the few cases in which k is not equal to a power of two a separate integrator is used as a constant multiplier. A constant k is placed in the Y register and the ΔY increment is connected to the ΔX input resulting in an output of $\Delta Z = k \Delta Y$.

An incremental computer can either be programmed to approximate the solution of a differential equation or, as is more common in present day practice, to solve a difference equation which may in some cases approximate a differential equation. In either case, the programming is accomplished by connecting the ΔZ outputs to the proper ΔX and ΔY inputs such that the resulting integrator configuration is constrained to solve the proper equation.

Two approaches to performing incremental computation have been examined. Either word slice or bit slice processing may be used and each method has unique advantages and disadvantages. The bit slice processing procedure will be described first.

A bit slice associative memory is arranged such that a given bit of all the words in the memory is processed at a particular time. The proposed word structure for a bit slice processor which is to be used for incremental computation is shown in Figure 4-2. There are a total of eight different fields in the word. Four of these fields are used to store the Y and R registers and the ΔX and ΔY increments. Since it is generally required that an integrator accept several ΔY inputs, provisions for these multiple ΔY inputs are included in the ΔY field. As previously stated ternary increments are used thus requiring two bits for the storage of each ΔX and each ΔY . The following coding scheme is proposed for the two sets of increments.

<u>a</u>	<u>b</u>	<u>Δ</u>
0	0	0
0	1	+1
1	0	-1
1	1	not used

This method of scaling facilitates searching first for positive increments and then for negative increments or vice versa. It also facilitates modification of the processing order by the sign of the increment as will be described later.

The four remaining fields are somewhat unconventional and require a more detailed explanation of their purposes. The scaling control bits are used for scaling the various integrators. This is done by first searching the first bit location in the field for ones and then adding or subtracting an increment to all of the Y words which have a one in this position. If there is more than

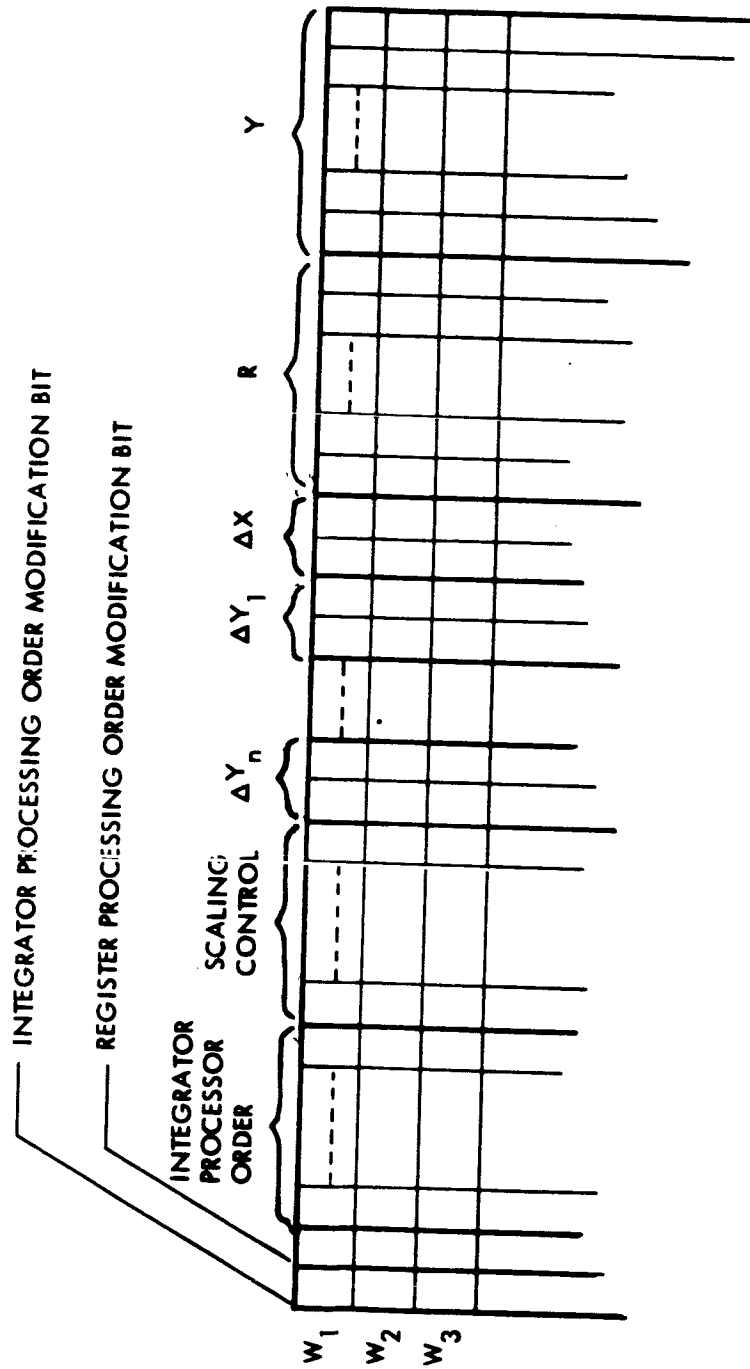


Figure 4-2. Memory Arrangement For a Bit Slice Associative Memory Used For Incremental Computation

one ΔY at any of these positions they must each be added sequentially. Then the second scaling bit location is searched for ones and the process is repeated until all of the scaling bits and ΔY bits have been searched. Notice that considerable time can be saved in this type of memory operation if a very rapid parallel search is available and if its results can be used to eliminate the succeeding steps whenever there are no matches in the search. Thus another desirable feature would be a rapid, parallel "no match" indicator which would sense this condition and immediately cause the processor to search the next bit location.

In some algorithms certain sets of integrators must be processed sequentially because of the nature of the difference equation being solved. The order of processing depends on whether the particular part of the equation being considered calls for the use of the old value of Y or the new value of Y. The old value of Y corresponds to the contents of a Y register that has not been updated during the present iteration period, while the new value of Y corresponds to the contents of a Y register that has been updated. The natural method of operation of this type of associative memory results in the consistent use of the old value of Y since the integrators are processed in parallel. Therefore, in order to use the new value of Y it is necessary to set aside one of the memory word fields to establish the order of integrators that must be processed sequentially. These bits are so indicated in Figure 4-2.

It is sometimes necessary to modify the processing order of sequentially processed integrators depending on the sign of the ΔX increment, or other criterion. This modification is obtained by means of logical operations performed on the processing order bits and on the ΔX bits. A single control bit has been added for this purpose. It permits a specified criterion to control the processing order. If more than one criterion is to be used additional control bits must be added. Some algorithms also require that the order of processing the Y and R registers be interchanged. An additional control bit has also been added for this purpose.

Many problems have been encountered in this attempt to use an associative memory to perform incremental computations. Up to this point all of these problems have been solved by the addition of extra bits to the memory word. Some of these bits contain data, others contain control information. One additional problem remains, however, that is not as easily solved. This is the problem of programming the processor to solve different sets of difference equations. Programming an incremental computer consists of merely directing the ΔZ overflows which are sensed at the end of each $Y + R$ addition and stored at the output of the memory (or internally if desired) to the proper ΔX and ΔY locations in other words of the memory. Of course if several ΔZ increments are to become ΔY increments for a given word they must each be directed to a separate ΔY storage location with the word. It has not been found possible to solve this problem by the addition of bits to the memory word. In fact, the only solution that is apparent at the present time consists of an external logic network which could be modified by a plug board arrangement to change the programming of the associative memory. One example of how this might be done is sketched in Figure 4-3.

The actual computation rate of an associative memory programmed to perform incremental computations depends on the search and write speeds of the processor and on the method of performing the computations. If it is assumed that a write operation requires approximately twice as much time as a search operation, a search speed in the range of 100 to 200 nanoseconds will probably be required to do the navigation and control computations. This figure was obtained by estimating the number of search operations (approximately 1700), and then calculating the required search speed for an assumed computation iteration rate of 3000 to 6000 per second. The resulting search speed appears to be within the range obtainable from a number of potentially useable memory elements.

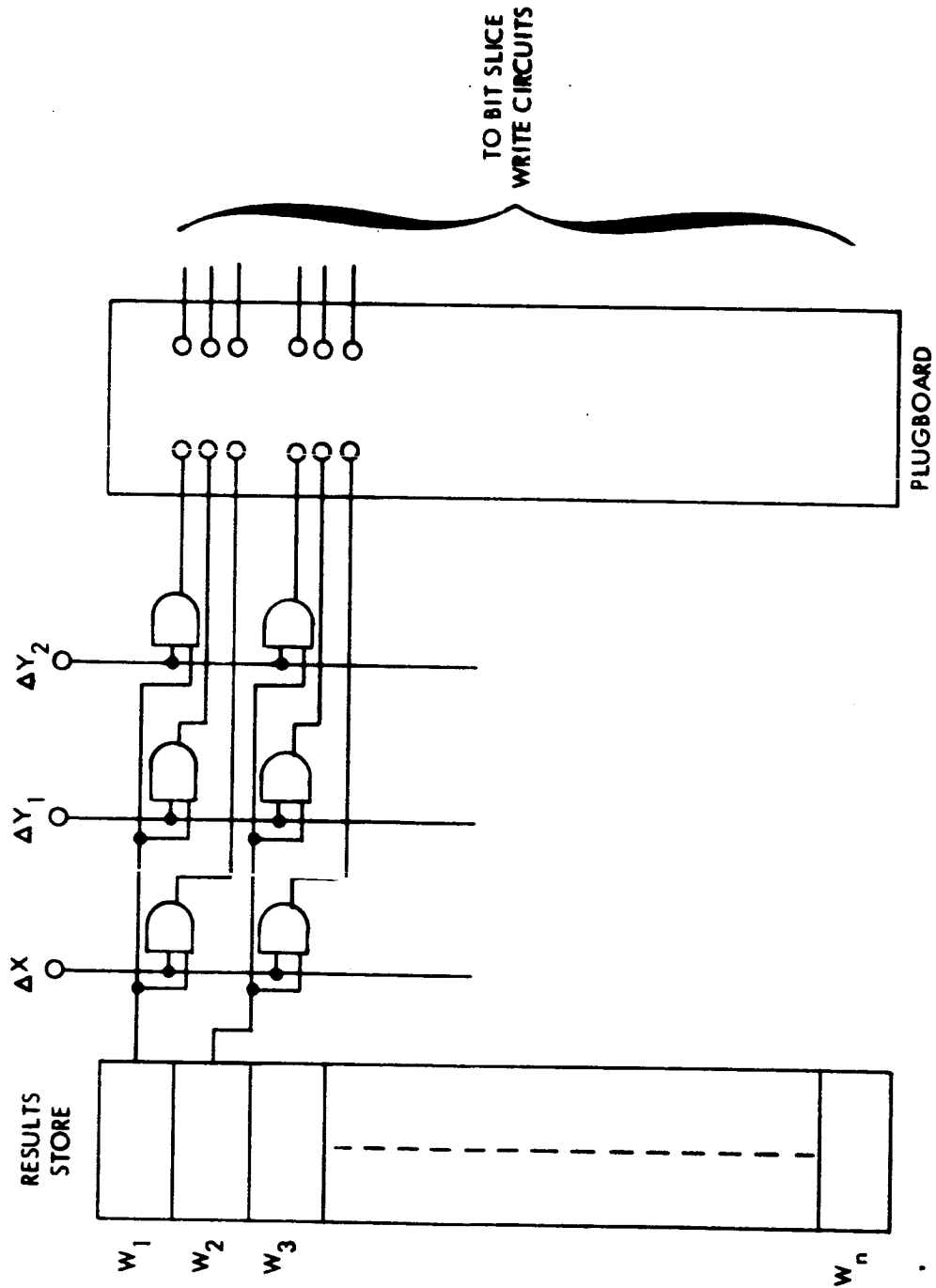


Figure 4-3. Plugboard Programming Arrangement For a Bit Slice Associative Memory Used For Incremental Computation

The previous discussion has pertained to a processor organization in which bit slice processing is employed. It is also possible to arrange the words in the memory in such a way that word slice processing is employed. The memory structure for an organization of this type is diagramed in Figure 4-4. In this organization integrator processing is inherently sequential while operations on a given word or pair of words are performed in a parallel-by-bit manner. Assuming a sequential processing of the integrators, in the order of their location in the memory, it is possible to use either the old value or the new value of Y by proper location of the integrators and by proper programming methods. It is difficult however, to vary the order of processing depending on the sign of one of the input or output increments or some other parameter. This would either require some complicated control logic in the sequencing and programming sections of the processor or else it would require the storage of the data and control bits for more than one integrator in a given column of the memory.

Updating of the Y register is performed by reading out the contents of the register along with the associated ΔY increments and the scaling bits, decoding the scaling bits, and then adding the ΔY increments to the Y register, one at a time, by means of a parallel adder. The R register is updated by reading out the R and Y registers and the ΔX increment, performing the required addition or subtraction, and rewriting the new contents of the R register. Programming is also a severe problem with this type of memory organization unless a two dimensional search is available. If this feature is available a set of rows could be set aside in the memory for programming purposes, two for each of the integrators. After a particular integrator is processed the corresponding programming row could be searched and the ΔZ could be written into each of the ΔY locations corresponding to a 1 in the first of the rows and into each of the ΔX locations corresponding to a 1 in the second of the two rows. A serious problem arises in that it would be difficult to know which of the ΔY locations to write into since one must be able to distinguish between an old ΔY which is to be replaced and a newer ΔY which was previously written during the same iteration of the memory.

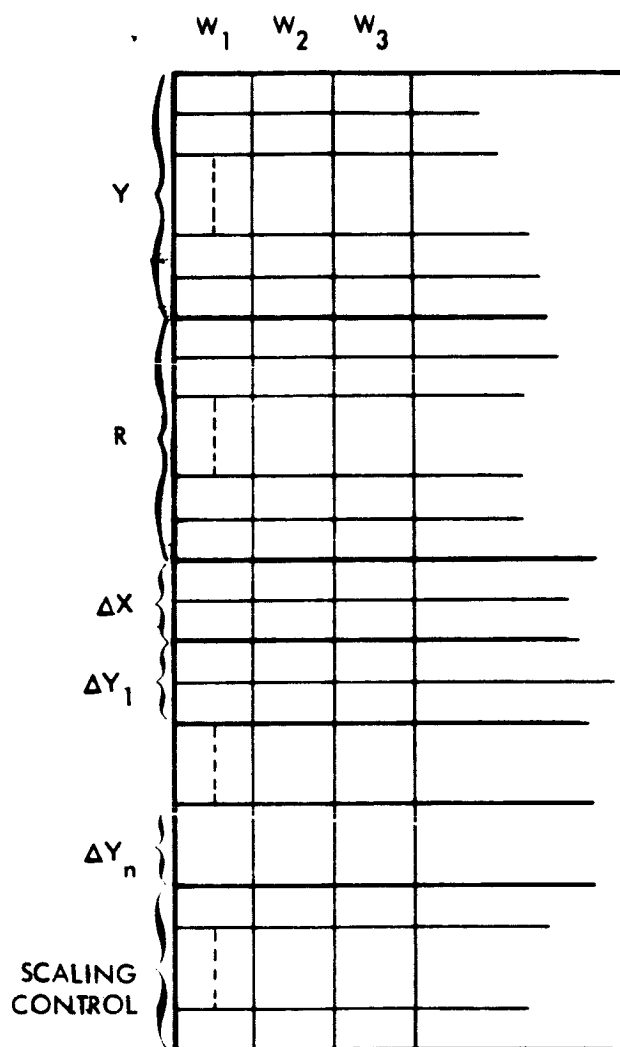


Figure 4-4. Memory Arrangement For a Word Slice Associative Memory Used For Incremental Computation

Each of these two methods of performing incremental computations in an associative processor has particular advantages and disadvantages. The first mechanization is most suitable for a computation that requires a large number of integrators, since the integrators are processed in more or less a parallel fashion while the bits are processed sequentially. Most of the necessary control functions can be readily performed by means of searches on special fields contained within the word. Programming, however, presents a problem and it appears that an external plugboard programming unit may be required. The second organization is more suitable for a computation requiring only a small number of integrators since the integrators are processed sequentially while operations on a given word are performed in parallel. All of the necessary control functions except for variation in the processing order and programming can be readily performed by means of special fields in the memory word. It might be possible to devise reasonable solutions to both of these problems if both a horizontal and a vertical search instruction were available. This brief investigation of the capabilities of an associative memory as an incremental computer has resulted in quite reasonable solutions to most of the problems involved in the outlook is certainly optimistic. It is expected that further investigation of the problems could result in feasible solutions to the remaining problems.

SOME CONSIDERATIONS IN THE ORGANIZATION OF AN ULTRA-RELIABLE ASSOCIATIVE MEMORY

An ultra-reliable associative memory is defined as one which can detect a failure and continue operating (possibly with reduced capabilities). In the event of a failure, the loss of the information in the failed cell or failed word is allowed.

When considering the problem of error detection, various approaches to organization of an associate processor can be divided into two groups — those without logic in the cell, and those with the logic distributed

throughout the memory. The techniques available for error detection are quite different in these two cases.

Consider first the associative memory with logic distributed throughout the array. Since the logic functions are performed locally, the error detection function must be distributed. Coding techniques can in general be excluded because of the local logic and the difficulty when masking is used. Duplex or majority logic at each cell are examples of methods that could be used. This would have an appreciable effect on the cost and power requirements.

In the type of associative memory without distributed logic, the problem of detecting errors becomes a less formidable one. Since all the masking and processing is performed external to the array, the array itself can be checked with a simple parity or with a slightly more complex residue check code. The logic external to the array can then be made ultra-reliable by using redundancy techniques. Since this redundancy is required only on a per word basis rather than a per cell basis, the effect in terms of cost and power would probably not be prohibitive.

Once a fault has been detected, some provision must be made for by-passing it. A fault along a word line would not appear to be a serious problem; that word must simply be excluded from search operations and must be avoided when loading the memory. However a fault along a bit line is a potentially more serious problem. One organizational feature that is worthwhile considering is a bi-directional capability. In this case operations can be carried out either along columns or row. Thus words can be stored either in the column or the rows of the array. If a fault were to occur such that one bit in every word could not operate properly, the entire contents of the array could be rotated 90° so that the fault effects a single word. This would imply that the memory is square (the number of bits per word equals the number of words) which would severely limit its application. Thus bi-directionality of operations should be considered a worthwhile feature for reliability only in special situations.

These are some initial thoughts on the subject of reliable associative memories. Some additional effort is planned in the remainder of the study.

SECTION V

SUMMARY OF RESULTS AND WORK TO BE DONE

The results of each of the tasks of the program have been described in previous sections. A brief summary of these results is given here and tasks remaining to be done are identified.

The computations required on an unmanned space vehicle were defined in terms of a set of computation descriptors. The most frequently used computation descriptor was the sum of products. The organization of an associative memory aimed at this computation is planned.

A number of general observations were made regarding the space exploration problem. First, there is a need for adaptive sampling of the scientific sensors as well as performance sensors. The capabilities of an associative memory fit this requirement sufficiently well to justify further investigation.

One of the most severe requirements on the on-board computing facility is for data compression. The capability of collecting large amounts of data along with a limited communications capability make it necessary to consider encoding and filtering techniques to assure that only pertinent data is sent. Compression of TV or radar pictures appears to be a particularly fruitful area and has been selected for further investigation.

Another type of requirement is that of incremental computation. This is used primarily in navigation and control of a lander vehicle. It has been shown that an associative memory can handle the requirement. The significant point of this result is that an associative memory, justified mainly for some other function, could handle this function during its relatively short duration and thus eliminate the special purpose hardware which would normally be furnished.

Another somewhat indirect observation concerning the on-board computing facility is the reliability problem. This has generated two subtasks, one of which is

partially completed. First, to achieve the ultra-reliability that the computing facility must have, it is assumed that it will be a multi-processor system. In such a system associative memories have been shown to be applicable to a number of the control functions. Therefore, this application will be given some attention. Also, the possibility of a centralized associative memory in the system places very stringent reliability requirements on it. Hence, some thought has been given to the organization of an ultra-reliable associative memory.

The emphasis in the remainder of the study will be on the special applications identified above. Also to be done, however, are two other tasks -- Task 4 - Evaluation of Associative Techniques for Space Exploration Computation, and Task 5 - Device Recommendations (see page 1-1). The first of these will use the outputs of Task 2 - the Survey of Organizational Approaches and Task 3 - The Examination of Special Applications and Approaches. Each approach will be evaluated in terms of its effectiveness on the computation requirements defined in Task 1. The final task, the device recommendations, will be made by making use of the results of the device survey. Device recommendations will be made by considering the results of the evaluation and the status or confidence level of applicable devices. It is expected that a small number of devices can be singled out as being particularly promising for space exploration requirements.